SARI-Tutorial: SAR Data Analysis, Signal Processing and Machine Learning using Python with Examples on Mangrove Mapping

Charlie Marshak, Marc Simard, Tien-Hao Liao, Michael Denbina
Jet Propulsion Laboratory, California Institute of Technology

December 20, 2010

Abstract

This is a tutorial held at the Prince of Songkla University, Phoket Campus. It is a collection of Jupyter Notebooks to demonstrate open-source tools to analyze SAR data with applications to Mangrove mapping.

1_Downloading_Data

December 5, 2019

1 Downloading Data

All the data in this tutorial has been downloaded for you. This page is a reference of some websites/open source tools/tutorials. We gratefully and humbly acknowledge JAXA for the use of ALOS-2 data to generate a coherence image over Mondah, Gabon generated using ISCE2.

We will reference image repositories throughout the tutorial wherever applicable. In this tutorial, we concentrate on images from the L-band SAR sensor from ALOS/PALSAR. L-band sensors are popular for forest analysis due to their long wavelengths and ability to penetrate canopy cover for additional information. ALOS-1 was a satellite that was active from 2007-2010 that had the L-band PALSAR-1 sensor and ALOS-2 is its successor with the PALSAR-2 sensor. There are numerous other satellite SAR sensors including Sentinel-1 (C-band) and UAVSAR (airborne L-band).

This list below was made relatively quickly. Hope this gets you started. Most of the links will direct you to NASA operated data repository, which are open but require a login. New users can register here.

2 SAR

2.1 ALOS-1

All the products below are for backscatter images that have been radiometrically terrain corrected.

1. Alaska Site Facility
   - ASF Search (formerly called the "Vertex")
     - A tutorial on how to download data from the ASF - simply navigate to "Event Recording" on the page; you will need to give permission to download the application Adobe Connect.
   - HyP3
     - this tools actually makes it easy for you to obtain coherence images (derived from pairs), interferograms, etc.
     - To do one-time processing of a site, you have to contact the ASF - you have monthly quotas.
2. JAXA Mosaics. You have to register first and then downloading is free. This provides global annual RTC mosaics at 25 m scale! *Warning*: known geocoding issues that appear has pixel/half-pixel shifts from year to year or when comparing to other SAR images.

2.2 ALOS-2

This is currently not open - we do not provide raw data that you would download - only the coherence product obtained via ISCE2.

However, there is a recently announced agreement so that a large portion of the radar data can be made public.

I want to emphasize how important (and exciting, at least for SAR) this is for forest monitoring and biomass analysis. L-band is extremely powerful for forest studies because it is able to penetrate the canopy. Additionally, the short repeat pass time of ALOS-2 (14 days!) makes a lot of the interferometric products equally interesting and valuable for large and insightful studies, even with zero-spatial baseline. We are truly grateful to JAXA for providing Simard's lab access to this data for large regional studies.

2.3 UAVSAR

1. UAVSAR data.
   - "Stacks" are for inteferograms
     - can process using the python library Kapok - here is an excellent Kapok tutorial.
     - ISCE2 can also process these stacks using there insarApp framework. Here is the related `xml` file - see more on ISCE2 below.
   - "PolSAR" data is for backscatter analysis and decompositions. Must be RTC-ed. HyP3 allows you to process this using the proprietary software gamma.
2. Marc Simard's UAVSAR data over Boreal forest in the Laurentides that has been processed can be dowloaded.

2.4 Sentinel-1

1. ESA's OpenHub
2. Alaska Site Facility Vertex
3. AWS Repo and related viewers (might cost $$ to request/download data) found here.

3 Optical

3.1 Landsat

1. Earth Explorer
2. Hansen Mosaics for Forest Analyses - first and last images are very useful.
   - Mosaics

4 Mangrove Extents

The global mangrove watch provides global mangrove extents good for reference or training a higher quality regional model. You have to send a form and the extents will be emailed to you almost instantly.

5 Processing Tools

5.1 ISCE2

ISCE2 is for SAR interferometry (see the references therein). We will use one interferometric products, namely the coherence, but the full range of what can be accomplished with SAR data is astounding e.g. geological deformation within centimeters of accuracy using ALOS-1. There are a plethora of tools that build on the products generated from ISCE2 such as mintpy, but this is far beyond the scope of this tutorial.

We used ISCE2 to process Alos-2 coherence data over Gabon. We note for mac and linux that there is a conda install! That means you can make your virtual environment and then install ISCE2 - the previous install instructions are atrocious.

If you are interested in learning more, here are some notebooks illustrating ISCE2 use cases for various SAR sensors including how to generate interferograms and coherence from ALOS-1 here.

There is quite a learning curve for this software. I would recommend obtaining an account for HyP3 and downloading data that is processed using ISCE2 or gamma on the cloud and inspecting the products to see if they are useful for your analysis.

5.2 SNAP

SNAP is meant to process all Sentinel data. This includes RTC-ing SAR images - fortunately, it read not just from Sentinel data. This excellent, detailed tutorial (sections 3 - 5) by Dr. Marc Simard details how to RTC SAR raw ALOS-1 images, which is highly relevant for using the tools discussed in this tutorial should an RTC image not be directly available.

5.3 QGIS

QGIS is a great open-source raster/vector viewer. Get to know it!

5.4 Google Earth Engine

Very powerful tool for obtaining remote sensing images, many of them processed and ready for analysis.

Here are some nice jupyter from AGU 2017 and related video resources from Tyler Erickson. There are tons of examples on the earth engine github as well.

5.5 Google Earth Pro

Google Earth Pro is a desktop application that has some nice functionality such as Google Satellite basemap and time lapse for validating change areas. Here) is a video tutorial for the latter.

6 Selected Tutorials

There are too many to list. Many more can be found at awesome-sar. Here are some highlights.

- SAR handbook on comprehensive methodologies for forest monitoring and biomass estimation. Has tutorials with sample data and code written by domain experts. Has a section on Mangrove mapping by Dr. Marc Simard.

- UNAVCO tutorials on remote sensing including many InSAR tutorials using ISCE2, GIAnT, etc.
- SAR tomography with video instruction.
- A tutorial on the application of convolutional neural nets for Hi-resolution optical data.

2_GIS

December 5, 2019

[1]:
```python
import sys
sys.path.append('../utils')
from rio_tools import reproject_arr_to_match_profile, get_cropped_profile
import rasterio
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from tqdm import tqdm
```

The main goal of this notebook is to illustrate how to coregister geospatial SAR products for environmental analysis and monitoring.

0.1 Rasterio

We are going to illustrate how to use `rasterio` for Global Information System mapping (GIS, for short) of raster data. `Rasterio` is a popular python library for manipulating georeferenced rasters. This library is very well maintained and constantly being improved. Additionally, there is healthy activity related to `rasterio` on GIS stack-exchange and an excellent quickstart tutorial. Not everything can be covered here, but more details and information can easily be found searching the `rasterio` github, rasterio documentation, and the GIS stack-exchange.

1 Raster data (GeoTIFF)

Scenario: we have a stack of SAR images and we want to use these images to do some form of environmental analysis such as change detection.

We need to be able to have a single reference frame so that:

1. Each image has the same size
2. The images have the same resolution.
3. The pixel position (i, j) refers to the same position on the map for *all images* in our stack.

Frequently, these stipulations mean that all the images in are stack are *coregistered*. In the `data` directory, we have two images that are not coregistered. Not only do the images have different different coverage areas, but different resolutions and sizes. View them yourselves in QGIS.

All the images we use in this tutorial are geotiff. Geospatial rasters not only stores the raster image, but also the geospatial metadata such as the image extents, resolution, etc so they can be easily converted when needed. There are numerous other raster formats, all but `netcdf` is supported in `rasterio`.

2 Inspecting the Data

Let's get the filepaths from the data directory.

[2]:
```
data_dir = Path('data')
tifs = list(data_dir.glob('*.tif'))
tifs
```

[2]:
```
[PosixPath('data/alos_coh_hh.tif'),
 PosixPath('data/alos_backscatter_hh.tif'),
 PosixPath('data/alos_backscatter_hv.tif'),
 PosixPath('data/alos_coh_hv.tif')]
```

We can select the files that are backscatter using python string functions.

[3]:
```
backscatter_tifs = sorted(list(filter(lambda path: 'backscatter' in path.name,␣
 ↪tifs)))
backscatter_tifs
```

[3]:
```
[PosixPath('data/alos_backscatter_hh.tif'),
 PosixPath('data/alos_backscatter_hv.tif')]
```

Same for coherence.

[4]:
```
coherence_tifs = sorted(list(filter(lambda path: 'coh' in path.name, tifs)))
coherence_tifs
```

[4]:
```
[PosixPath('data/alos_coh_hh.tif'), PosixPath('data/alos_coh_hv.tif')]
```

2.1 Image Metadata

We open files using `rasterio`. Note the "pythonic" way of opening files - within the `with` statement block the raster object is read and automatically closed once the end of the block is reached.

[5]:
```
with rasterio.open(coherence_tifs[0]) as ds:
    coh_hh = ds.read()
    coh_profile = ds.profile
```

The relevant metadata for rasterio is contained in a profile as a python dictionary. For those familiar with `gdal`, this is a concise form of `gdalinfo`.

[6]:
```
coh_profile
```

[6]:
```
{'driver': 'GTiff', 'dtype': 'float32', 'nodata': nan, 'width': 500, 'height':
500, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform':
Affine(0.0002777777777777778, 0.0, 9.436944444444444,
       0.0, -0.0002777777777777778, 0.6111111111111109), 'tiled': False,
'interleave': 'band'}
```

[7]:
```
print('This is the crs: ', coh_profile['crs'])
print('This is the origin and resolution data: ', coh_profile['transform'])
print('This is the datatype of the raster: ', coh_profile['dtype'])
print('This is how many bands are in the raster', coh_profile['count'])
```

```
This is the crs:  EPSG:4326
This is the origin and resolution data:  | 0.00, 0.00, 9.44|
| 0.00,-0.00, 0.61|
| 0.00, 0.00, 1.00|
This is the datatype of the raster:  float32
This is how many bands are in the raster 1
```

The "transform" above is an object from the `Affine` library. It's very handy when manipulating geodata - we will briefly discuss this affine transformation in a later section of this notebook.

2.1.1 Image Formats

Spatial dimensions and channels can be ordered in various ways. Let our image have spatial dimensions $M \times N$, that is M pixels in the vertical axis and N in the horizontal.

- BIP - band interleaved by pixel - (# of bands/channels) $\times M \times N$ - this is how `gdal` and `rasterio` store raster data
- BSQ - band sequential - $M \times N \times$ (# of bands/channels) - how `numpy` stores raster data
- BIL - band interleaved by line - $M \times$ (# of bands/channels) $\times N$ - not releveant but included for completeness.

Indeed, rasterio's images are BIP as illustrated below:

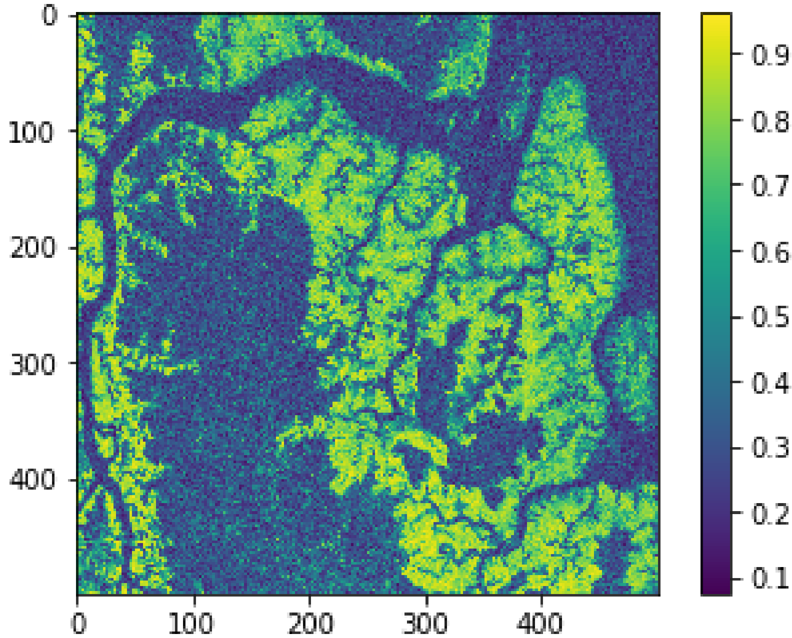
[8]:
```
print('Rasterio shape: ', coh_hh.shape)
```

```
Rasterio shape:  (1, 500, 500)
```

We can view the single band in matplotlib as a matrix $M \times N$, but BIP images are not understood by matplotlib. Below, we select the first band to display.

[9]:
```
plt.imshow(coh_hh[0, ...])
plt.colorbar()
```

[9]: `<matplotlib.colorbar.Colorbar at 0x7fa041157860>`



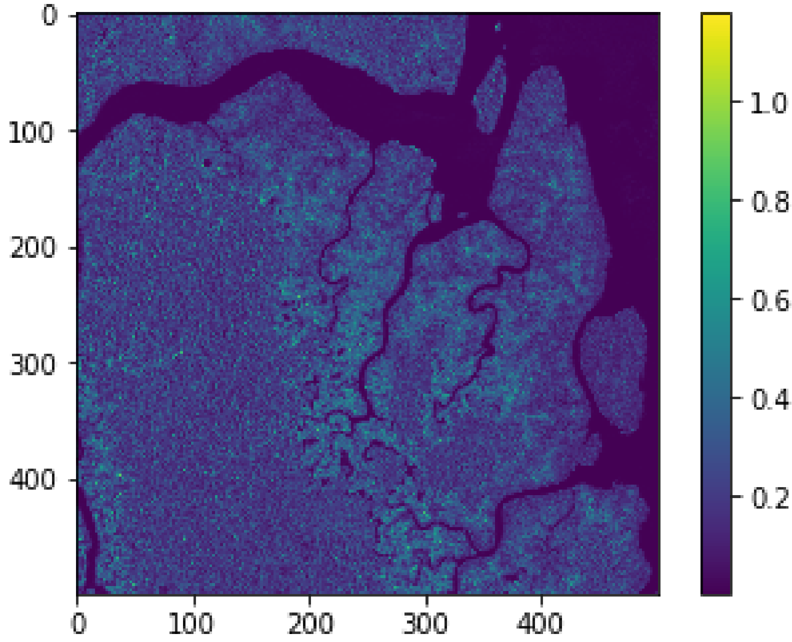
3 Reprojecting to Backscatter Reference Frame

The backscatter images and the coherence images have different reference frames. We select the backscatter reference frame and will reproject the coherence images into that reference frame - we could have easily done the reverse namely reprojecting the backscatter images to the coherence reference frame.

[10]:
```
with rasterio.open(backscatter_tifs[0]) as ds:
    backscatter_hh = ds.read()
    backscatter_profile = ds.profile
```

[11]:
```
plt.imshow(backscatter_hh[0, ...])
plt.colorbar()
```

[11]: <matplotlib.colorbar.Colorbar at 0x7fa04119f048>



What's different in the profiles below?

[12]:
```
coh_profile, backscatter_profile
```

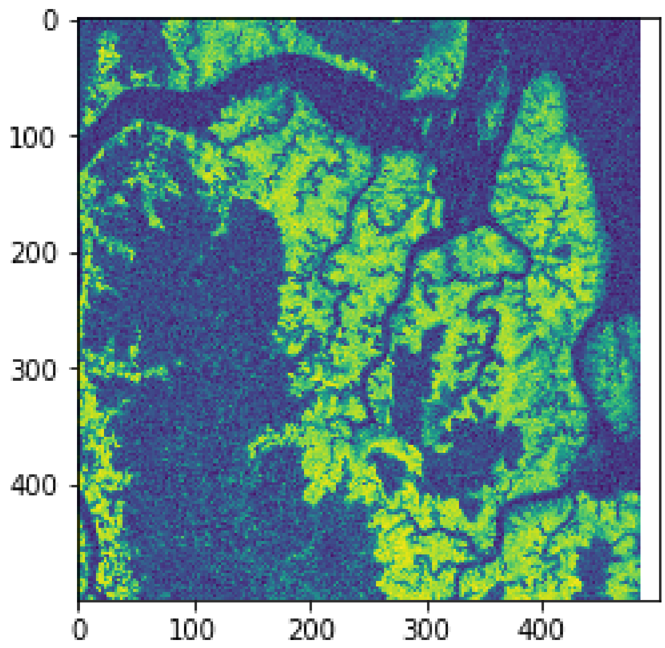
[12]:
```
({'driver': 'GTiff', 'dtype': 'float32', 'nodata': nan, 'width': 500, 'height':
500, 'count': 1, 'crs': CRS.from_epsg(4326), 'transform':
Affine(0.0002777777777777778, 0.0, 9.436944444444444,
        0.0, -0.0002777777777777778, 0.6111111111111109), 'tiled': False,
'interleave': 'band'},
 {'driver': 'GTiff', 'dtype': 'float32', 'nodata': 0.0, 'width': 500, 'height':
500, 'count': 1, 'crs': CRS.from_epsg(32732), 'transform': Affine(30.0, 0.0,
549563.875,
        0.0, -30.0, 10067362.0), 'tiled': False, 'compress': 'packbits',
'interleave': 'band'})
```

We have a function that shortens this from the docs.

[13]:
```
coh_hh_r, coh_hh_profile = reproject_arr_to_match_profile(coh_hh,
                                                          coh_profile,
                                                          backscatter_profile,
                                                          resampling='bilinear')
```

[14]:
```
plt.imshow(coh_hh_r[0,...])
```

[14]:
```
<matplotlib.image.AxesImage at 0x7fa041504908>
```



We now write this image to disk.

[15]:
```
out = Path('out')
out.mkdir(exist_ok=True)
with rasterio.open(out/coherence_tifs[0].name, 'w', **coh_hh_profile) as ds:
    ds.write(coh_hh_r)
```

4 Excercise 1

Reproject the HV coherence image into the backscatter image.

[16]:
```
## Answer

with rasterio.open(coherence_tifs[1]) as ds:
    coh_hv = ds.read()
    coh_profile = ds.profile
```

[17]:
```
## Answer

coh_hv_r, coh_hv_profile = reproject_arr_to_match_profile(coh_hv, coh_profile,␣
 ↪backscatter_profile, resampling='bilinear')
with rasterio.open(out/coherence_tifs[1].name, 'w', **coh_hv_profile) as ds:
    ds.write(coh_hv_r)
```

5 Creating A nodata Mask

We need a consistent nodata mask for the images we reprojected above.

[18]:
```
reprojected_tifs = list(out.glob('*.tif'))
reprojected_tifs
```

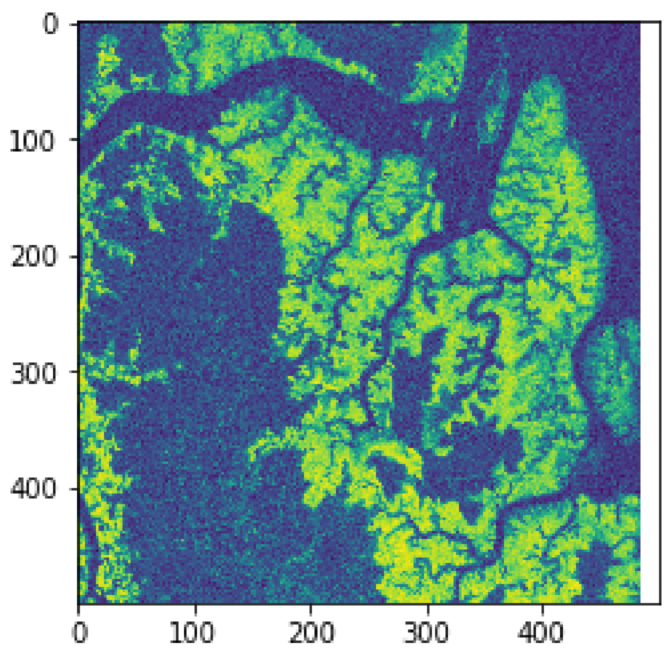
[18]: [PosixPath('out/alos_coh_hh.tif'), PosixPath('out/alos_coh_hv.tif')]

[19]:
```
tifs_for_cube = reprojected_tifs + backscatter_tifs
```

[20]:
```
def open_one_single_band_image(path):
    with rasterio.open(path) as ds:
        band = ds.read(1)
    return band

bands = list(map(open_one_single_band_image, tifs_for_cube))
```

[21]:
```
plt.imshow(bands[0])
```

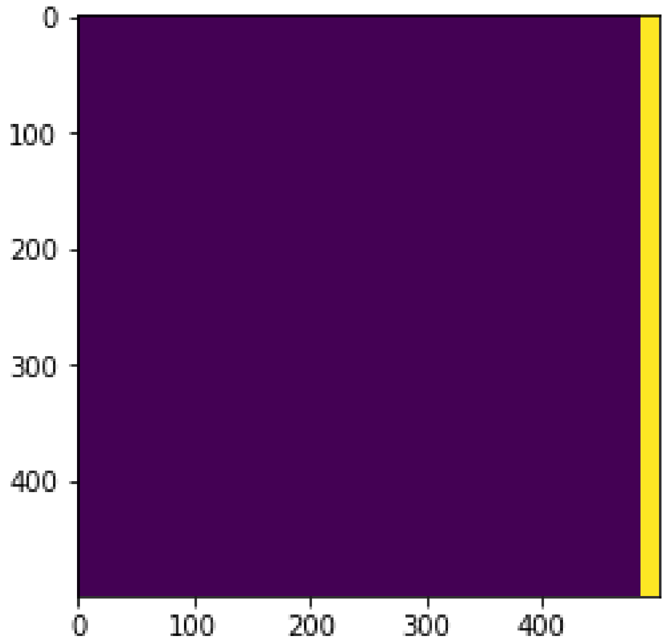
[21]: <matplotlib.image.AxesImage at 0x7fa0309176d8>



[22]:
```
# Method 1

nodata_mask = np.zeros(bands[0].shape, dtype=bool)
for k in range(len(bands)):
```

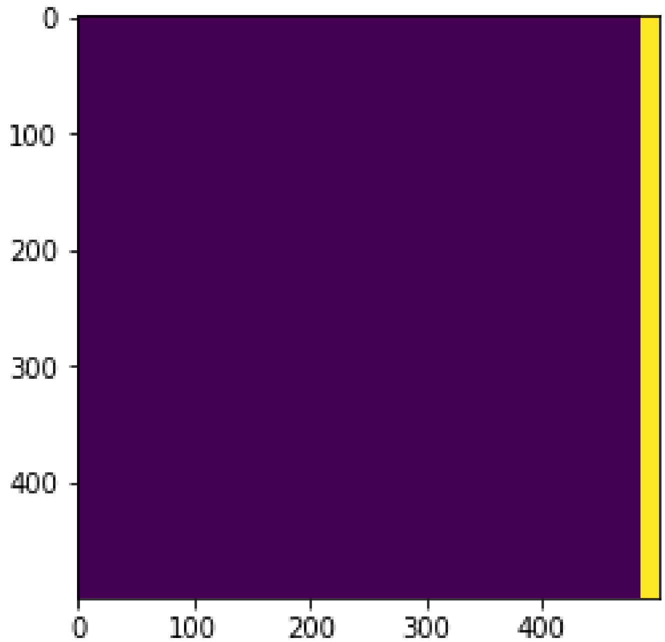
```
    # the "|" means or entrywise
    nodata_mask = (nodata_mask | np.isnan(bands[0]))
plt.imshow(nodata_mask)
```

[22]: <matplotlib.image.AxesImage at 0x7f9fe81ef5f8>



[23]:
```
# Method 2 - more functional

nodata_bands = list(map(np.isnan, bands))
nodata_mask = np.logical_or.reduce(nodata_bands)
plt.imshow(nodata_mask)
```

[23]: <matplotlib.image.AxesImage at 0x7f9fe02cfe10>

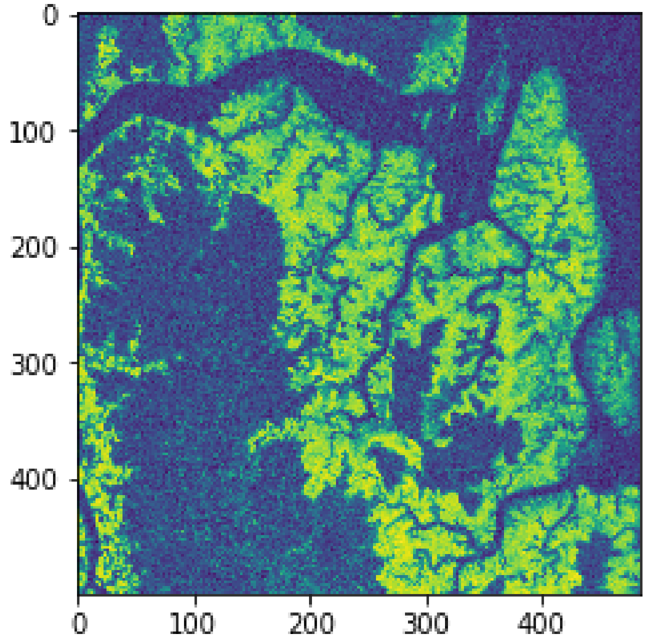


6 Cropping an Image

Here is how you can crop the images and remove the nodata areas using `get_cropped_profile`.

[24]:
```
max_x = np.argmax(nodata_mask[0,:])
sy = np.s_[:]
sx = np.s_[:max_x]
plt.imshow(bands[0][sy, sx])
```

[24]: `<matplotlib.image.AxesImage at 0x7fa0505f0d30>`



[25]:
```
reference_profile = get_cropped_profile(backscatter_profile, sx, sy)
reference_profile
```

[25]:
```
{'driver': 'GTiff', 'dtype': 'float32', 'nodata': 0.0, 'width': 484, 'height':
500, 'count': 1, 'crs': CRS.from_epsg(32732), 'transform': Affine(30.0, 0.0,
549563.875,
       0.0, -30.0, 10067362.0), 'tiled': False, 'compress': 'packbits',
'interleave': 'band'}
```

[26]:
```
def crop_one(band):
    band_r, _ = reproject_arr_to_match_profile(band,
                                               backscatter_profile,
                                               reference_profile,
                                               resampling='nearest')
    return band_r
```

[27]:
```
bands_c = list(map(crop_one, bands))
len(bands_c)
```

[27]: 4

[28]:
```
bands_c[-1].shape
```

[28]: (1, 500, 484)

Write these bands to a file.

[29]:
```
cropped_dir = Path('cropped')
cropped_dir.mkdir(exist_ok=True)

def write_one(data):
    band, name = data
    with rasterio.open(cropped_dir/name, 'w', **reference_profile) as ds:
        ds.write(band)
    return cropped_dir/name
```

[30]:
```
names = list(map(lambda x: x.name, tifs_for_cube))
data = list(zip(bands_c, names))
list(map(write_one, data))
```

[30]:
```
[PosixPath('cropped/alos_coh_hh.tif'),
 PosixPath('cropped/alos_coh_hv.tif'),
 PosixPath('cropped/alos_backscatter_hh.tif'),
 PosixPath('cropped/alos_backscatter_hv.tif')]
```

7 A Digression into Transforms

Transforms control the *extents* and *resolution* of our coordinate reference frame. They are encoded as affine transformations and are implemented in python using the affine library, which includes routines for evaluating the affine transformations on pixel coordinates (i, j).

The affine transformation T: pixel coordinates $\to$ georferenced coordinates as a matrix multiplication, namely, $T \cdot (i, j)$ is the georeference point of the pixel in position (x, y) on the map. For example, we can perform this using the coherence images lat, lon reference frame from above.

[31]:
```
with rasterio.open(coherence_tifs[0]) as ds:
    trans = ds.transform
```

[32]:
```
from geopy import distance

x0, y0 = trans * (0, 0)
x1, y0 = trans * (1, 0)
x0, y1 = trans * (0, 1)

# these are the georeferenced coordinates
x0, y0, x1, y1
```

[32]:
```
(9.436944444444444, 0.6111111111111109, 9.437222222222221, 0.6108333333333331)
```

`x0, y0` is the upper left corner of the image.

8 Exercise 2

Use the transform above to determine how big the upper left resolution frame is.

Use `distance.distance` from the geopy library as explained here.

Note, their function takes tuples that are in (lat, lon), i.e. (y, x).

[33]:
```
## Answer

distance.distance((y0, x0), (y1, x0)).meters
```

[33]: 30.71511168703892

[34]:
```
## Answer

distance.distance((y0, x0), (y0, x1)).meters
```

[34]: 30.9203336956487

9 Acknowledgements

We gratefully and humbly acknowledge JAXA for the use of ALOS-2 data to generate a coherence image over Mondah, Gabon generated using ISCE2.

We are grateful to the ASF for providing easy access to ALOS-1 radiometrically and terrain corrected tiles over Mondah and Ko Panyi.

A portion of this research was performed at the Jet Propulsion Laboratory, California Institute of Technology. US Government Support Acknowledged.

3_Backscatter

December 5, 2019

[1]:
```
import sys
sys.path.append('../utils')

import rasterio
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from tqdm import tqdm
from sklearn.cluster import KMeans
```

The main goals of this notebook are:

1. To demonstrate how to inspect backscatter images.
2. Introduce the use K-Means unsupervised clustering algorithm to classify pixels in SAR images. For those unfamiliar with K-means - here is a nice video explaining the algorithm in detail. We will be using the sklearn library, which has very good documentation about the algorithm and its use.

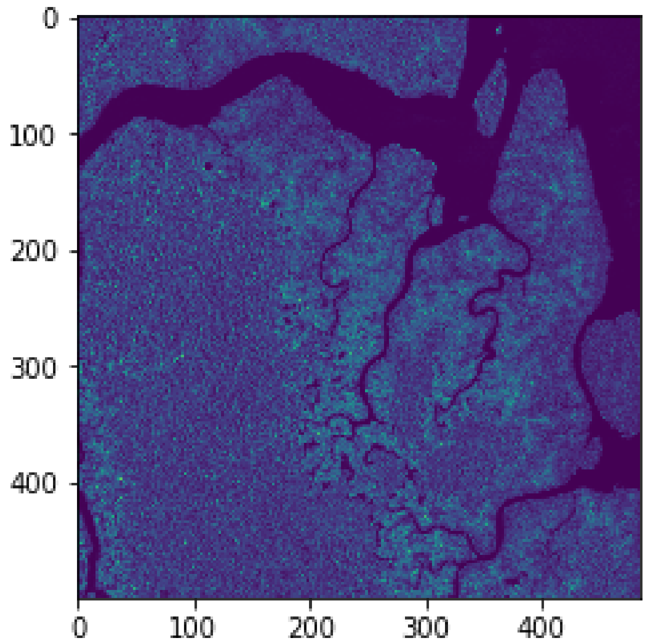
1 Inspecting the Data

[2]:
```
data_dir = Path('../data')
backscatter_tifs = sorted(list(data_dir.glob('./*backscatter*.tif')))
backscatter_tifs
```

[2]:
```
[PosixPath('../data/alos_backscatter_hh.tif'),
 PosixPath('../data/alos_backscatter_hv.tif')]
```

[3]:
```
with rasterio.open(backscatter_tifs[0]) as ds:
    hh = ds.read(1)
plt.imshow(hh)
```

[3]:
```
<matplotlib.image.AxesImage at 0x7fc9703bda90>
```



Above, the images are in linear γ^0 and have been radiometrically terrain corrected (RTC). We will not have the time to discuss the details of this processing of backscatter images. Fortunately, the products viewed above can be obtained for download directly from HyP3 or the ASF vertex.

If you cannot obtain RTC images directly, there are ways of processing lower level radar data using open source software such as ISCE2. Here is an example for Sentinenel-2. Additionally, such correction can be accomplished using SNAP as discussed in this excellent, detailed tutorial by Dr. Marc Simard.

For this tutorial, this is what we need to know:

1. γ^0 is typically used for forest and biomass analysis
2. Radiometric and terrain correction tries to remove incidence angle dependence and layovers. A more detailed account of this correction can be found in David Small's paper.

2 Decibels

As microwave remote sensing is an active remote sensing, it is frequently advantageous to view these images in decibels. Frequently, the dynamic range of a backscatter image is reported in dB. Also, noise in linear γ is multiplicative. In dB, however, the becomes additive as

$$\log(I \cdot \varepsilon) = \log(I) + \log(\varepsilon)$$

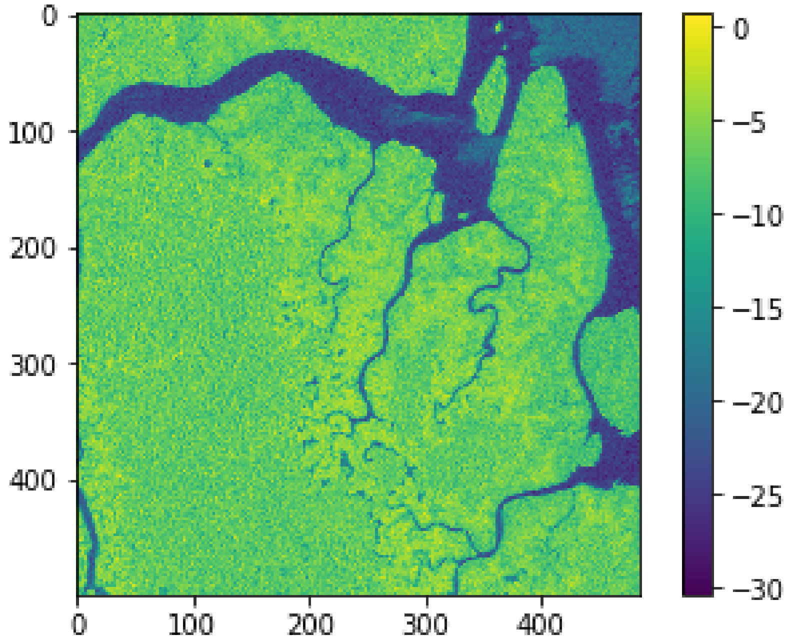
where I is the image and ε is the noise.

Finding the dB image requires only one line of code.

[4]:
```
hh_db = 10 * np.log10(hh)
```

[5]:
```
plt.imshow(hh_db)
plt.colorbar()
```

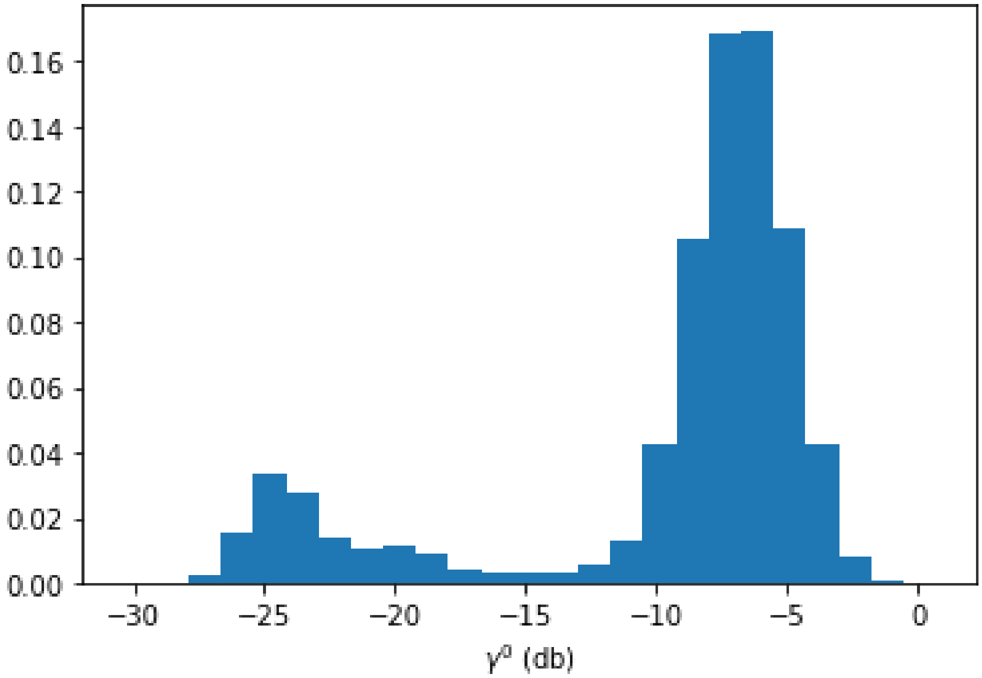
[5]: <matplotlib.colorbar.Colorbar at 0x7fc9a1b59c50>



We can look at the histograms of backscatters in dB.

[6]:
```
plt.hist(hh_db.ravel(),
         bins=25,
         density=True)
plt.xlabel('$\gamma^0$ (db)')
```

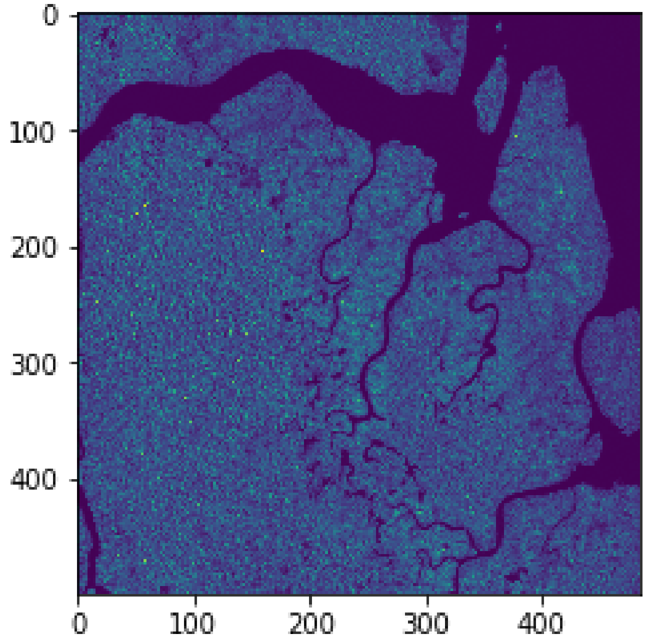
[6]: Text(0.5, 0, '$\\gamma^0$ (db)')



Let's now do the sam for HV polarization.

[7]:
```
with rasterio.open(backscatter_tifs[1]) as ds:
    hv = ds.read(1)
plt.imshow(hv)
```

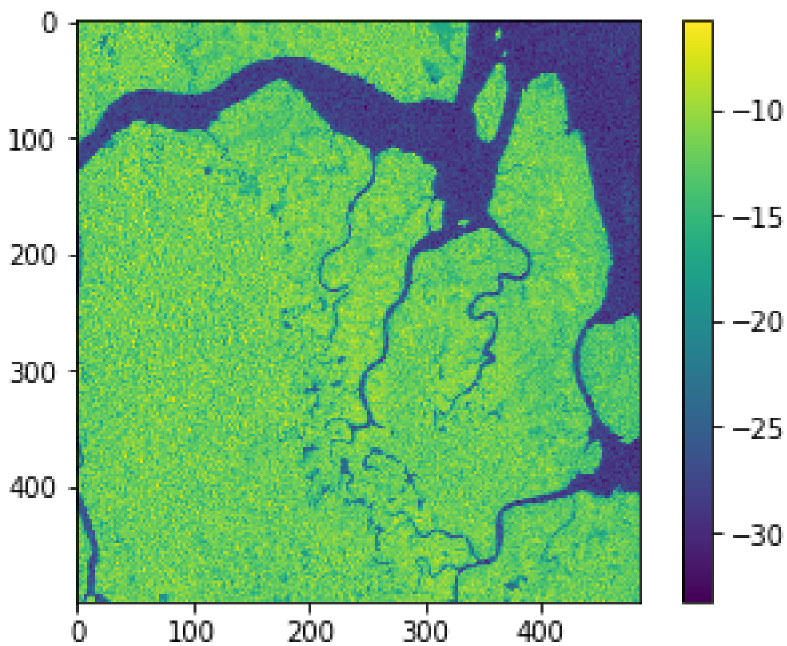
[7]: `<matplotlib.image.AxesImage at 0x7fc960191320>`




```
[8]: hv_db = 10 * np.log10(hv)
```

```
[9]: plt.imshow(hv_db)
     plt.colorbar()
```

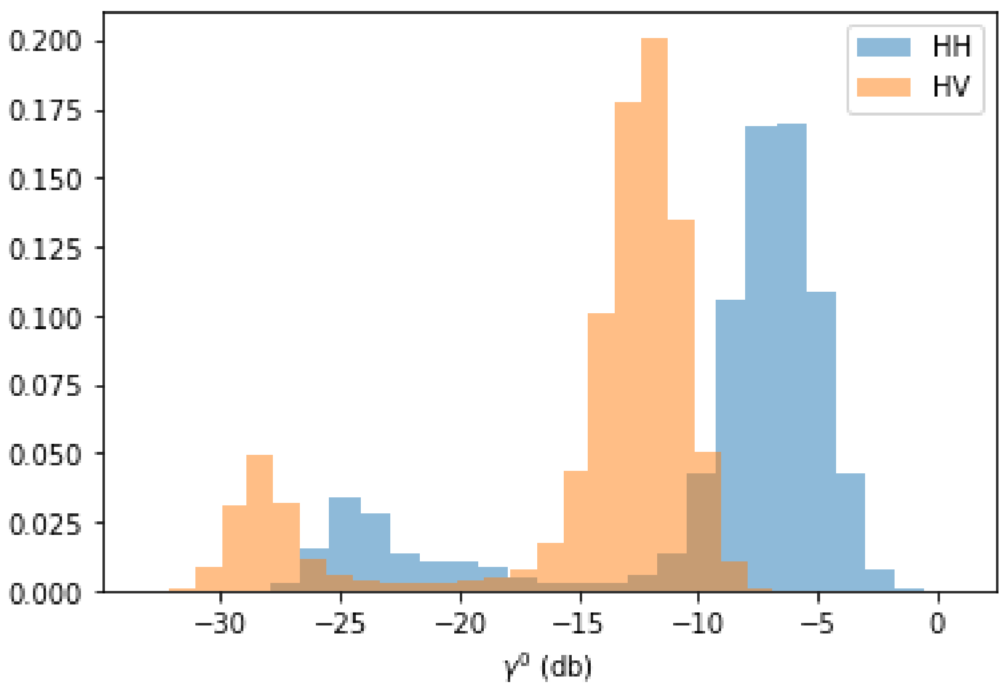
[9]: <matplotlib.colorbar.Colorbar at 0x7fc9a1f626d8>



We can compare the two images in a histogram.

[10]:
```
plt.hist(hh_db.ravel(),
         bins=25,
         density=True,
         label='HH',
         # alpha is the transparency
         alpha=.5)
plt.hist(hv_db.ravel(),
         bins=25,
         density=True,
         label='HV',
         # alpha is the transparency
         alpha=.5)
plt.xlabel('$\gamma^0$ (db)')
plt.legend()
```

[10]: <matplotlib.legend.Legend at 0x7fc9501fbba8>



3 Classifying Pixels

Classification is an important portion of statistics and machine learning. Roughly, given a dataset

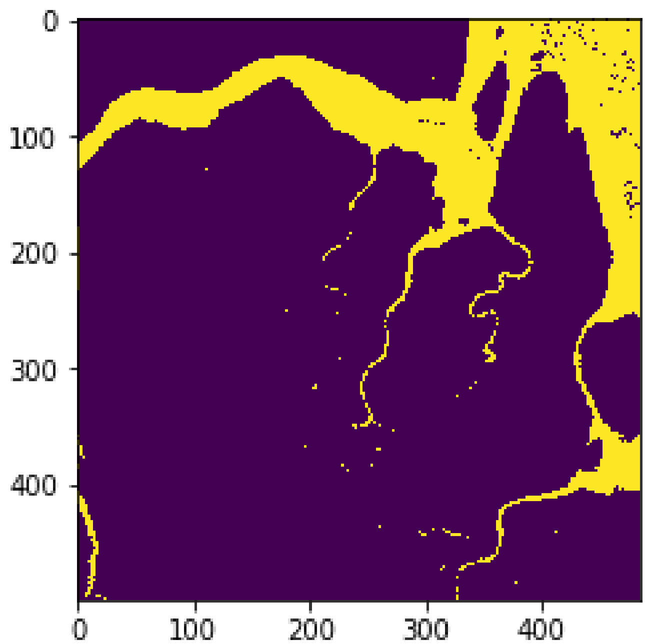
$\mathcal{D}$, classification wishes to learn the mapping f into some set of finite labels L so that $f : \mathcal{D} \to L$. In our case, our dataset $\mathcal{D}$ consists of the pixel backscatter values that is $\mathcal{D} = \{\gamma^0_{ij} \; : \; i = 1, ..., M, \; j = 1, ...N\}$ and our labels L could be landcover classes such as mangroves, water, etc.

We will focus on K-means because this is unsupervised. This simply means we do not use pre-existing knowledge of the label mapping to improve performance. We have chosen an unsupervised approach because it is very easy to apply and a good way to get a sense of what information is present in the SAR image.

Let's start with the easiest scenario, namely, looking only at HH backscatter. We know that water tends to have low HH backscatter in SAR image (discussed here). So, we can start looking at thresholds to determine classes of 0 (land) and 1 (water).

[11]:
```
plt.imshow((hh_db < -18).astype(int))
```

[11]: <matplotlib.image.AxesImage at 0x7fc96026ca90>



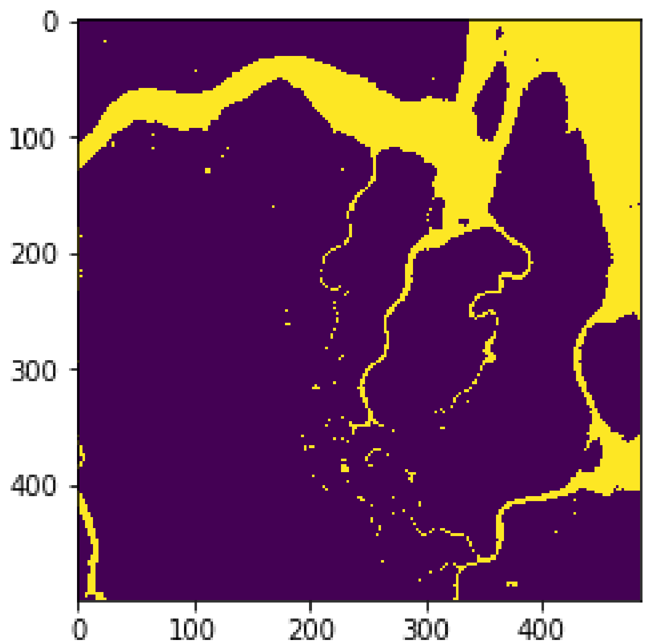
We can also use the distribution of the HH backscatter to determine this automatically with K-means.

[12]:
```
model = KMeans(# How many classes we want - in this case, just water and land
               n_clusters=2,
               # this ensures the models output is the same each time,
               random_state=0)

model.fit(hh_db.reshape((-1,1)))
```

[12]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)

[13]:
```
classes = model.labels_.reshape(hh_db.shape)
plt.imshow(classes)
```

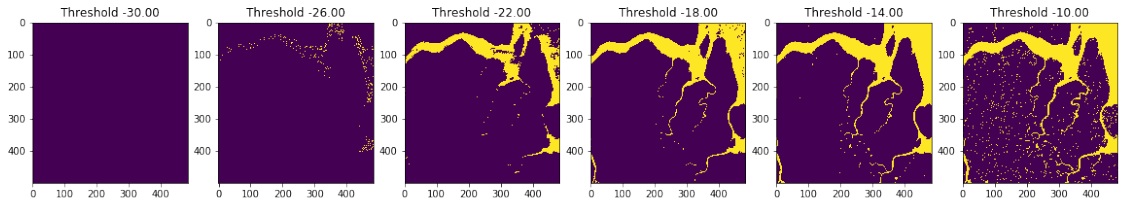
[13]: <matplotlib.image.AxesImage at 0x7fc9611b7390>



4 Excercise 1

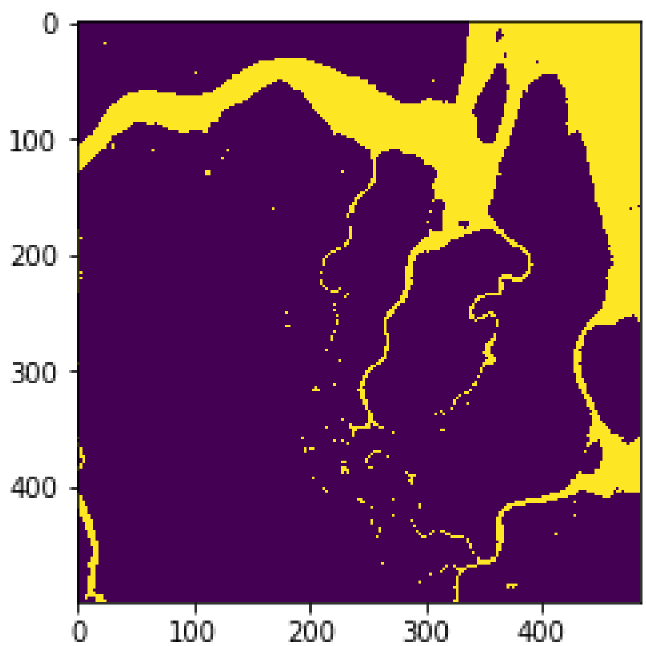
Plot thresholds from -30 to -10 for some integer stepsize and see how the land/water threshold partitions the image. Write one these classifcations as a GeoTIFF and see how it looks in QGIS using a basemap.

[14]:
```
## Answer
N=6
thresh_arr = np.linspace(-30, -10, N)
fig, ax = plt.subplots(1, N, figsize=(20, 10))
for ax, thresh in zip(ax.ravel(), thresh_arr):
    ax.imshow(hh_db < thresh)
    ax.set_title(f'Threshold {thresh:1.2f}')
```



[15]:
```python
## Answer

X = (hh_db < -15).astype(np.uint8)
plt.imshow(X)
```

[15]: <matplotlib.image.AxesImage at 0x7fc9602547b8>



[16]:
```python
with rasterio.open(backscatter_tifs[0]) as ds:
    profile = ds.profile

profile['nodata'] = None
profile['dtype'] = 'uint8'
with rasterio.open('threshold.tif', 'w', **profile) as ds:
    ds.write(X, 1)
```

5 Excercise 2

Apply K-means to pixels with *HH and HV* and divide the image into *3* classes. How does it do?

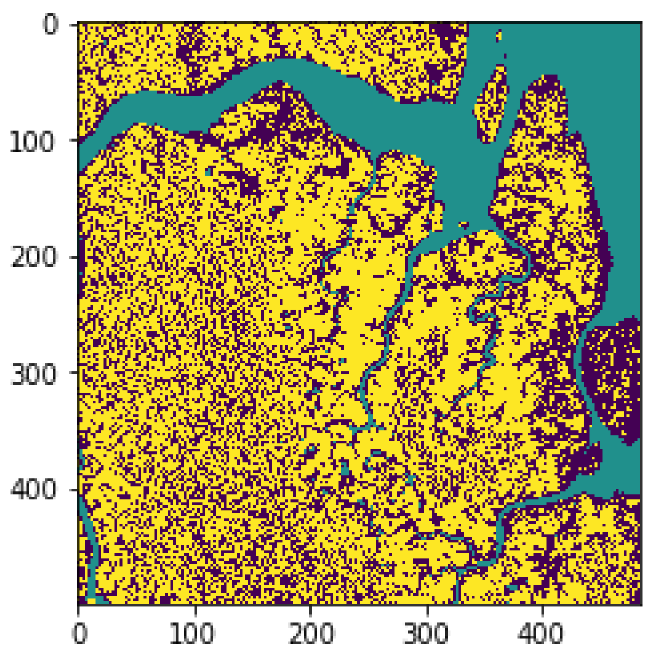
Write your classes to a file and inspect the results.

[17]:
```
## Answer

model_2 = KMeans(n_clusters=3,
                 random_state=0)
X_train = np.zeros((hh.size, 2))
X_train[:, 0] = hh_db.ravel()
X_train[:, 1] = hv_db.ravel()
model_2.fit(X_train)
```

[17]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

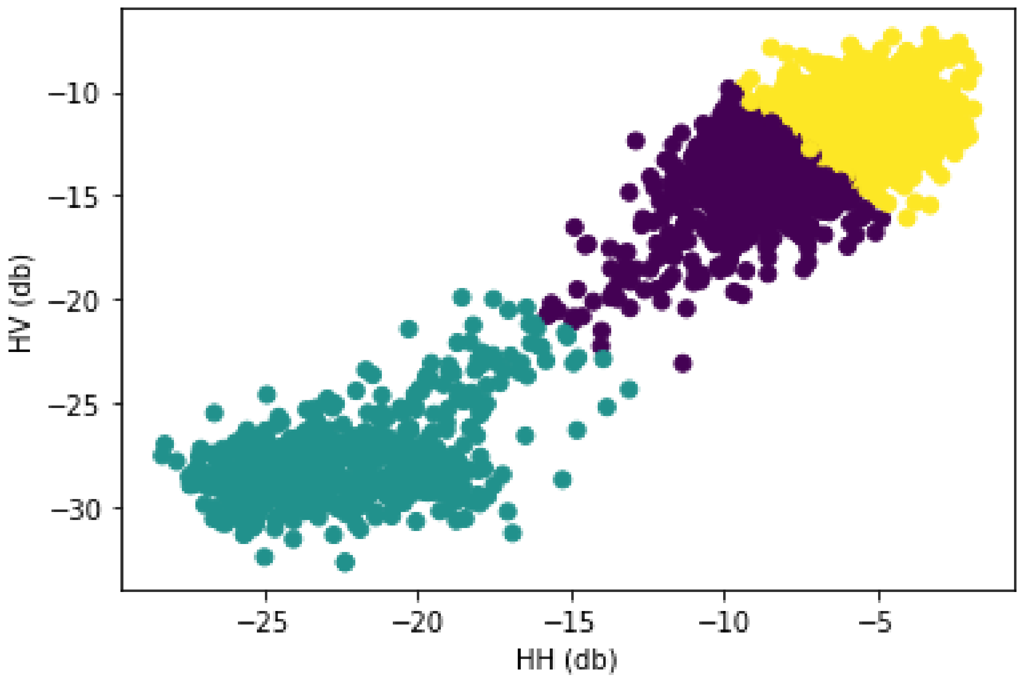
[18]:
```
## Answer

classes_2 = model_2.labels_.reshape(hh_db.shape)
plt.imshow(classes_2)
```

[18]:
```
<matplotlib.image.AxesImage at 0x7fc9708866d8>
```



[19]:
```
## Answer

N = 4000
indices = np.random.choice(np.arange(hh_db.size), N)
plt.scatter(hh_db.ravel()[indices], hv_db.ravel()[indices], c=classes_2.
 ↪ravel()[indices])
plt.xlabel('HH (db)')
plt.ylabel('HV (db)')
```

[19]: Text(0, 0.5, 'HV (db)')



[20]:
```
## Answer

with rasterio.open(backscatter_tifs[0]) as ds:
    profile = ds.profile

profile['nodata'] = None
profile['dtype'] = 'uint8'
with rasterio.open('kmeans.tif', 'w', **profile) as ds:
    ds.write(classes_2.astype(np.uint8), 1)
```

6 Acknowledgements

We gratefully and humbly acknowledge JAXA for the use of ALOS-2 data to generate a coherence image over Mondah, Gabon generated using ISCE2.

We are grateful to the ASF for providing easy access to ALOS-1 radiometrically and terrain corrected tiles over Mondah and Ko Panyi.

A portion of this research was performed at the Jet Propulsion Laboratory, California Institute of Technology.

4_Despeckling

December 5, 2019

[1]:
```
import sys
sys.path.append('../utils')
from nd_tools import lee_filter

import rasterio
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from tqdm import tqdm
from sklearn.cluster import KMeans
from skimage.restoration import denoise_tv_bregman
```

The goals of this notebook are:

1. Introduce two despeckling techiniques for backscatter images, namely the Lee Filter and Total Variation Denoising.
2. To inspect how such despeckling improves the classification we began in earlier notebooks.

1 Inspecting the Data

[2]:
```
data_dir = Path('../data')
backscatter_tifs = sorted(list(data_dir.glob('./*backscatter*.tif')))
backscatter_tifs
```

[2]:
```
[PosixPath('../data/alos_backscatter_hh.tif'),
 PosixPath('../data/alos_backscatter_hv.tif')]
```

[3]:
```
with rasterio.open(backscatter_tifs[0]) as ds:
    hh = ds.read(1)
    profile = ds.profile

with rasterio.open(backscatter_tifs[1]) as ds:
    hv = ds.read(1)
```

2 The Lee Filter

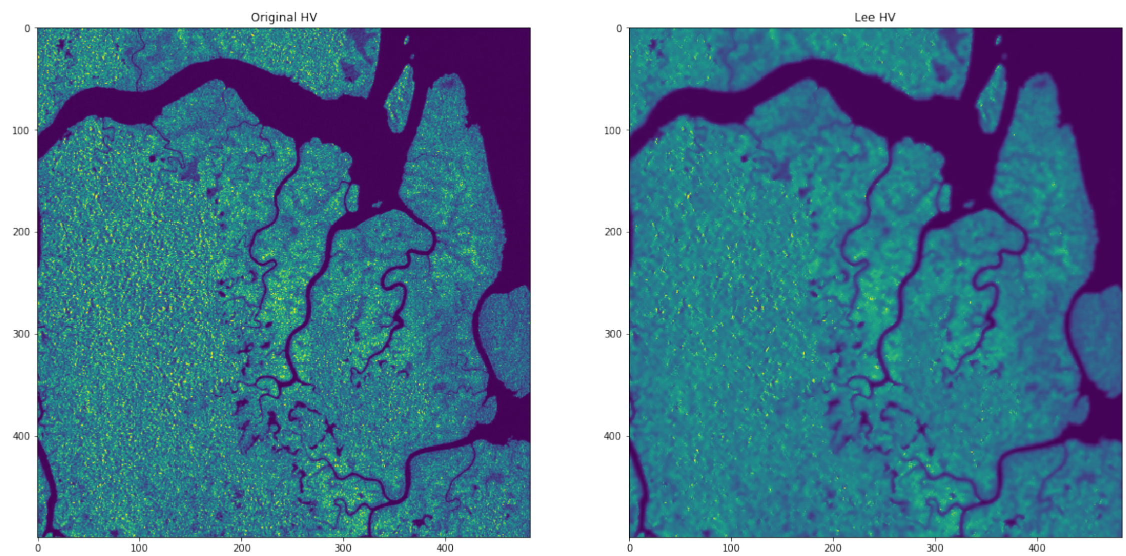
The Lee Filter we are using is from Stackoverflow. We compare the variance throughout the image to the variance within a $n \times n$ window. We move the image's pixelwise value closer to the mean within the window proportionally to how close the window's variance matches the image's variance. We have included the code in our utils.

We can compare the original image with the filtered image.

[4]:
```
hv_lee = lee_filter(hv, 5)
```

[5]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 10))
ax[0].imshow(hv, vmin=0, vmax=.15)
ax[0].set_title('Original HV')

ax[1].imshow(hv_lee, vmin=0, vmax=.15)
ax[1].set_title('Lee HV')
```

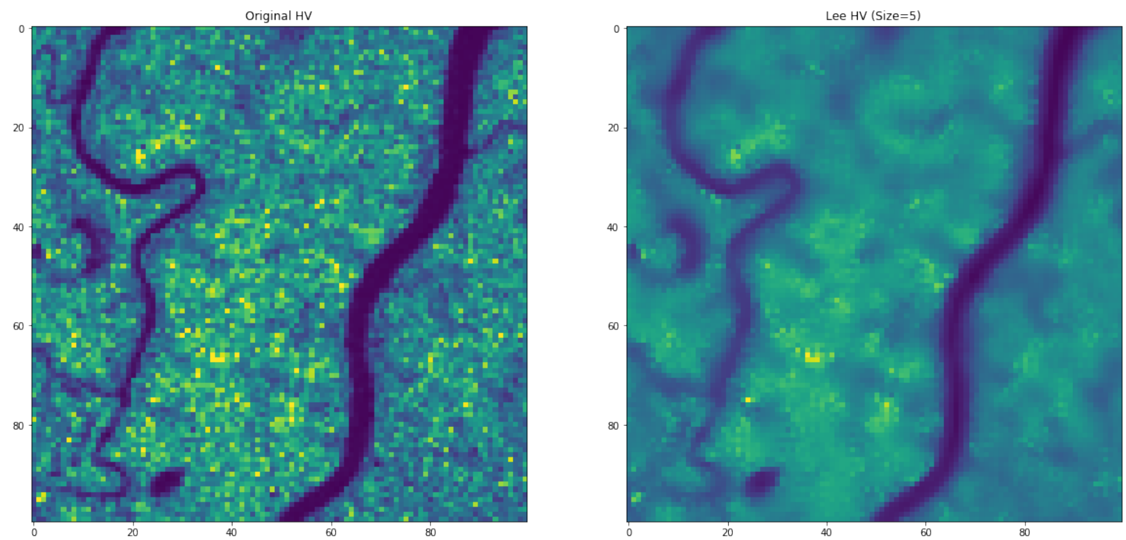
[5]: Text(0.5, 1.0, 'Lee HV')



Let's zoom in.

[6]:
```
s_index = np.s_[200:300, 200: 300]
fig, ax = plt.subplots(1, 2, figsize=(20, 10))

ax[0].imshow(hv[s_index], vmin=0, vmax=.15)
ax[0].set_title('Original HV')

ax[1].imshow(hv_lee[s_index], vmin=0, vmax=.15)
ax[1].set_title('Lee HV (Size=5)')
```

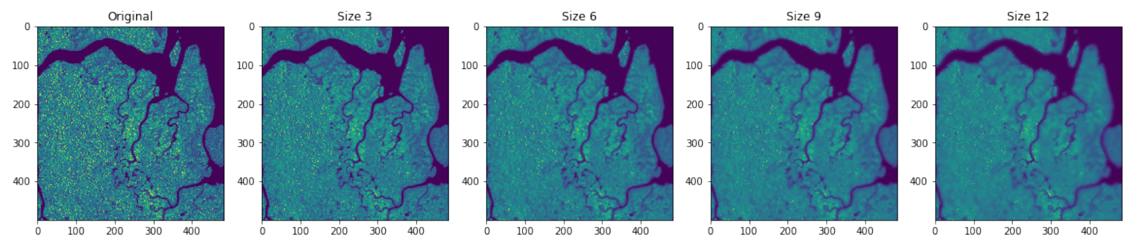
[6]: Text(0.5, 1.0, 'Lee HV (Size=5)')



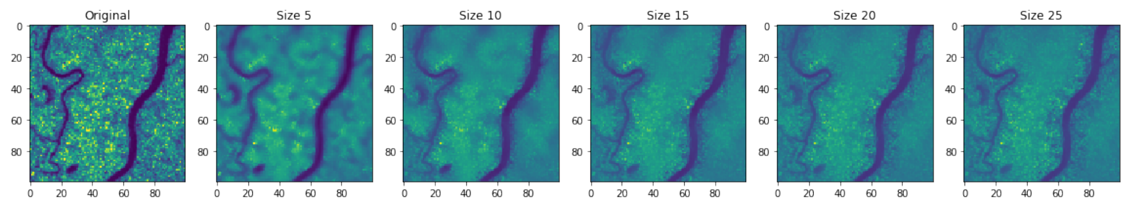
2.1 Window sizes and Lee Filters

Let's look at how the size of the window affects the image. Can you explain why larger window sizes apprears to increase the smoothing?

[7]:
```
sizes = [0] + list(range(3, 13, 3))
N = len(sizes)
fig, ax = plt.subplots(1, N, figsize=(20, 10))
for (ax, size) in zip(ax.ravel(), sizes):
    if size == 0:
        ax.imshow(hv, vmin=0, vmax=.15)
        ax.set_title('Original')
    else:
        hv_lee = lee_filter(hv, size)
        ax.imshow(hv_lee, vmin=0, vmax=.15)
        ax.set_title(f'Size {size}')
```



[8]:
```
sizes = [0] + list(range(5, 26, 5))
N = len(sizes)
fig, ax = plt.subplots(1, N, figsize=(20, 10))
for (ax, size) in zip(ax.ravel(), sizes):
    if size == 0:
        ax.imshow(hv[s_index], vmin=0, vmax=.15)
        ax.set_title('Original')
    else:
        hv_lee = lee_filter(hv, size)
        ax.imshow(hv_lee[s_index], vmin=0, vmax=.15)
        ax.set_title(f'Size {size}')
```



3 Total Variation Denoising

Total variation (TV) denoising refers to the total-variation regularizer $|\nabla f|_1$ to denoise an image with additive noise where we view an image as a function f from the image pixel domain Ω to $\mathbb{R}$. This regularizer for image denoising was first introduced in this seminal paper and a fast method for obtaining such regularized solutions was introduced here. Since, TV has become a very popular image processing technique particularly in remote sensing (just look at the number of titles with "total variation" in IGARSS every year!).

Here is nice tutorial on total variation of images with a discussion of the theory and its application. We use the implementation of total variation denoising found in scikit-image for which an example can be found here.

We will apply TV to SAR using the framework of "MuLOG" without debiasing since we are interested in classification and debiasing introduces a constant linear multiple to the γ^0 image which ultimately won't affect our classification.

We do the following: we transform the image to decibels so our noise is additive, denoise it with TV, and and then transform it back to linear units.

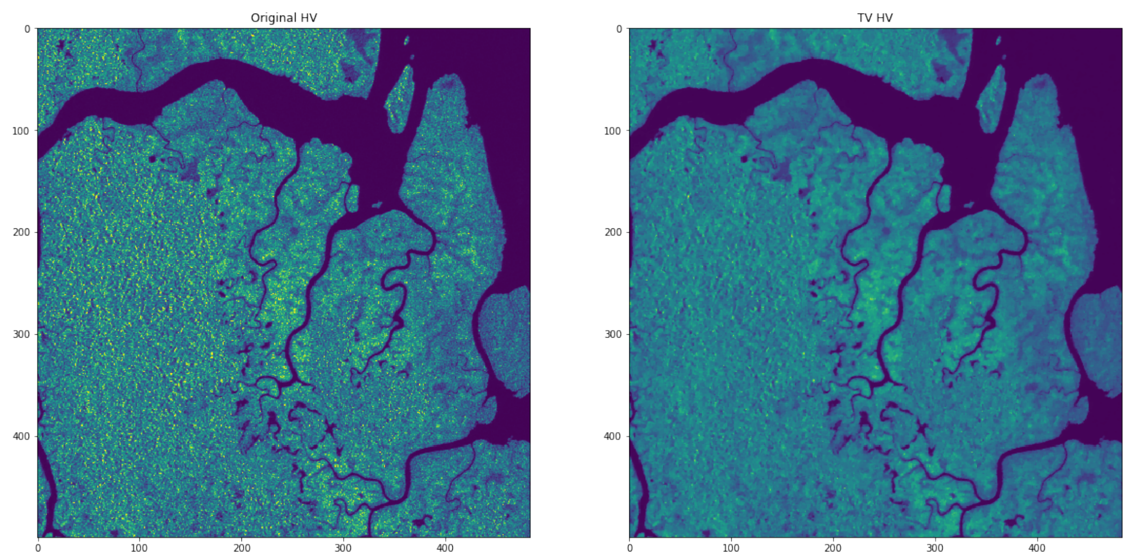
[9]:
```
def fwd(img):
    return 10 * np.log10(img)

def bwd(img):
    return 10**(img / 10)

def tv_denoise(img, weight):
```

```
    img_db = fwd(img)
    img_db_tv = denoise_tv_bregman(img_db, weight)
    img_tv = bwd(img_db_tv)
    return img_tv
```

[10]:
```
hv_tv = tv_denoise(hv, 1)
```

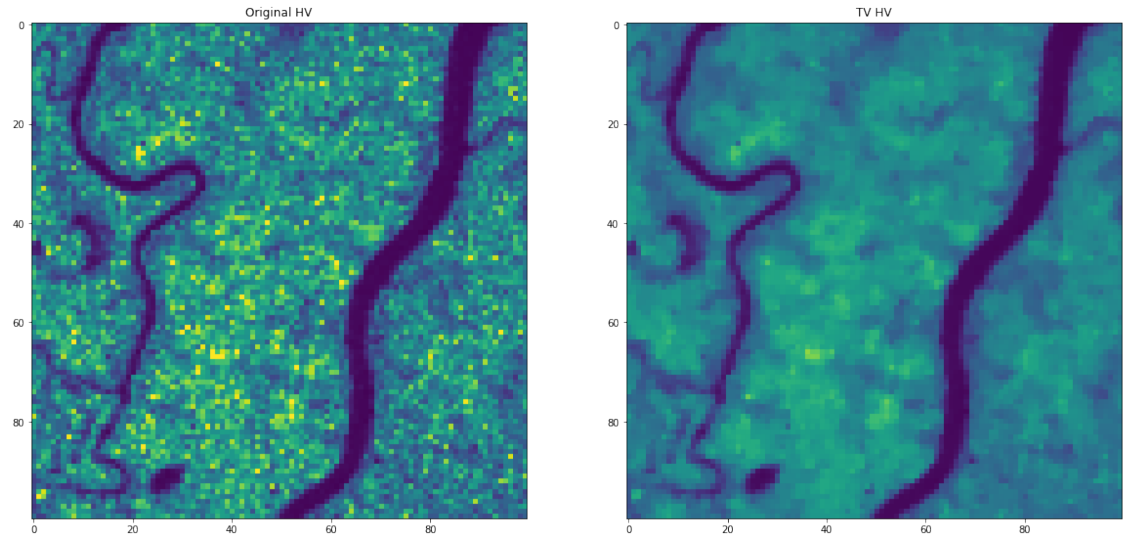
[11]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 10))
ax[0].imshow(hv, vmin=0, vmax=.15)
ax[0].set_title('Original HV')

ax[1].imshow(hv_tv, vmin=0, vmax=.15)
ax[1].set_title('TV HV')
```

[11]:
```
Text(0.5, 1.0, 'TV HV')
```



[12]:
```
s_index = np.s_[200:300, 200: 300]
fig, ax = plt.subplots(1, 2, figsize=(20, 10))

ax[0].imshow(hv[s_index], vmin=0, vmax=.15)
ax[0].set_title('Original HV')

ax[1].imshow(hv_tv[s_index], vmin=0, vmax=.15)
ax[1].set_title('TV HV')
```

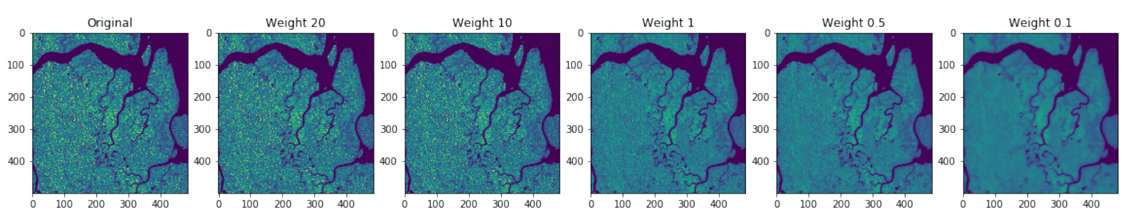
[12]:
```
Text(0.5, 1.0, 'TV HV')
```



4 Excercise 1

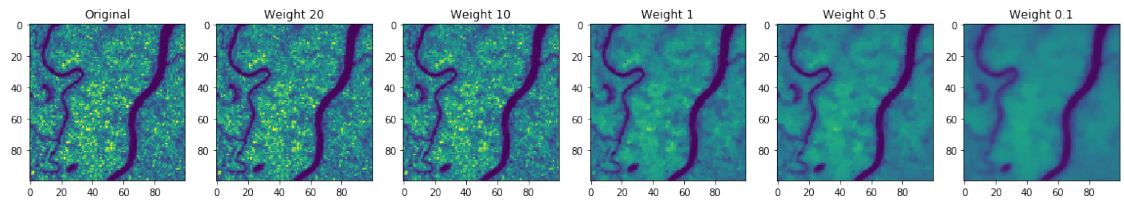
Compare weights of TV denoising for HV and HH images.

[13]:
```
## Answer
weights = [0] + [20, 10, 1, .5, .1]
N = len(weights)
fig, ax = plt.subplots(1, N, figsize=(20, 10))
for (ax, weight) in zip(ax.ravel(), weights):
    if weight == 0:
        ax.imshow(hv, vmin=0, vmax=.15)
        ax.set_title('Original')
    else:
        hv_tv = tv_denoise(hv, weight)
        ax.imshow(hv_tv, vmin=0, vmax=.15)
        ax.set_title(f'Weight {weight}')
```

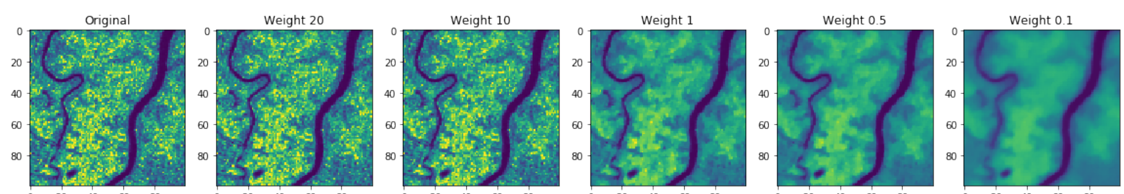


[14]:
```
## Answer
weights = [0] + [20, 10, 1, .5, .1]
N = len(weights)
```

```
fig, ax = plt.subplots(1, N, figsize=(20, 10))
for (ax, weight) in zip(ax.ravel(), weights):
    if weight == 0:
        ax.imshow(hv[s_index], vmin=0, vmax=.15)
        ax.set_title('Original')
    else:
        hv_tv = tv_denoise(hv, weight)
        ax.imshow(hv_tv[s_index], vmin=0, vmax=.15)
        ax.set_title(f'Weight {weight}')
```



[15]:
```
## Answer
weights = [0] + [20, 10, 1, .5, .1]
N = len(weights)
fig, ax = plt.subplots(1, N, figsize=(20, 10))
for (ax, weight) in zip(ax.ravel(), weights):
    if weight == 0:
        ax.imshow(hh[s_index], vmin=0, vmax=.5)
        ax.set_title('Original')
    else:
        hh_tv = tv_denoise(hh, weight)
        ax.imshow(hh_tv[s_index], vmin=0, vmax=.5)
        ax.set_title(f'Weight {weight}')
```



5 Writing our Denoisied Images to TIF

We'll select a weight of .5 and write the rasters to tif. We can then inspect these denoised images in QGIS and compare them to the original images.

[16]:
```
out_dir = Path('out')
out_dir.mkdir(exist_ok=True)

with rasterio.open(backscatter_tifs[0]) as ds:
    reference_profile = ds.profile
```

[17]:
```
def open_one(path):
    with rasterio.open(path) as ds:
        img = ds.read(1)
    return img

def tv_partial(band):
    # this looks like a good weight based on the above inspection
    return tv_denoise(band, .5)

def write_one(band, name):
    dest_path = out_dir/name
    with rasterio.open(dest_path, 'w', **reference_profile) as ds:
        ds.write(band, 1)
    return dest_path

def tv_writer(path):
    band = open_one(path)
    band_tv = tv_partial(band)
    print(band_tv.shape)
    dest_path = write_one(band_tv, path.name.replace('.tif', '_tv.tif'))
    return dest_path
```

[18]:
```
list(map(tv_writer, backscatter_tifs))
```

```
(500, 484)
(500, 484)
```

[18]:
```
[PosixPath('out/alos_backscatter_hh_tv.tif'),
 PosixPath('out/alos_backscatter_hv_tv.tif')]
```

6 Excercise 2

Use k-means with three classes on the denoised images What do you observe?

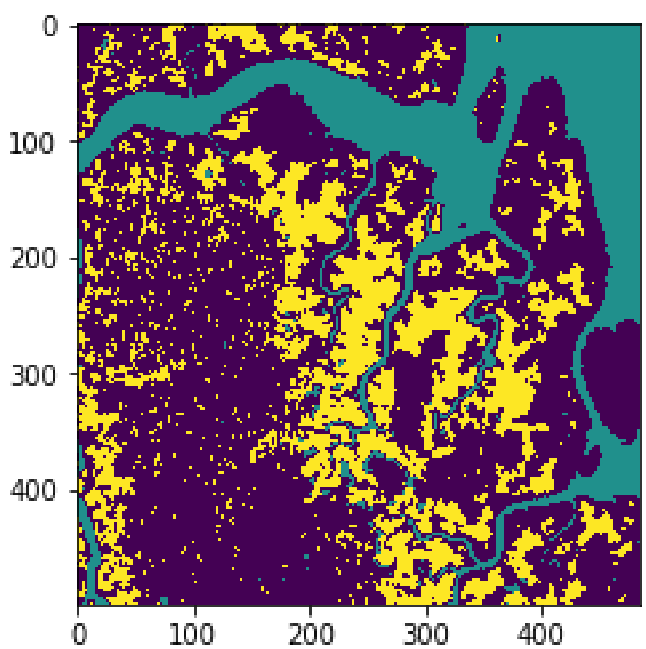
[19]:
```
## Answer

hh_tv = tv_denoise(hh, .5)
hv_tv = tv_denoise(hv, .5)
```

[20]:
```python
## Answer

model = KMeans(n_clusters=3,
               random_state=0)
X_train = np.zeros((hh.size, 2))
X_train[:, 0] = hh_tv.ravel()
X_train[:, 1] = hv_tv.ravel()
model.fit(X_train)
```

[20]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[21]:
```python
## Answer

classes = model.labels_.reshape(hh.shape)
plt.imshow(classes)
```

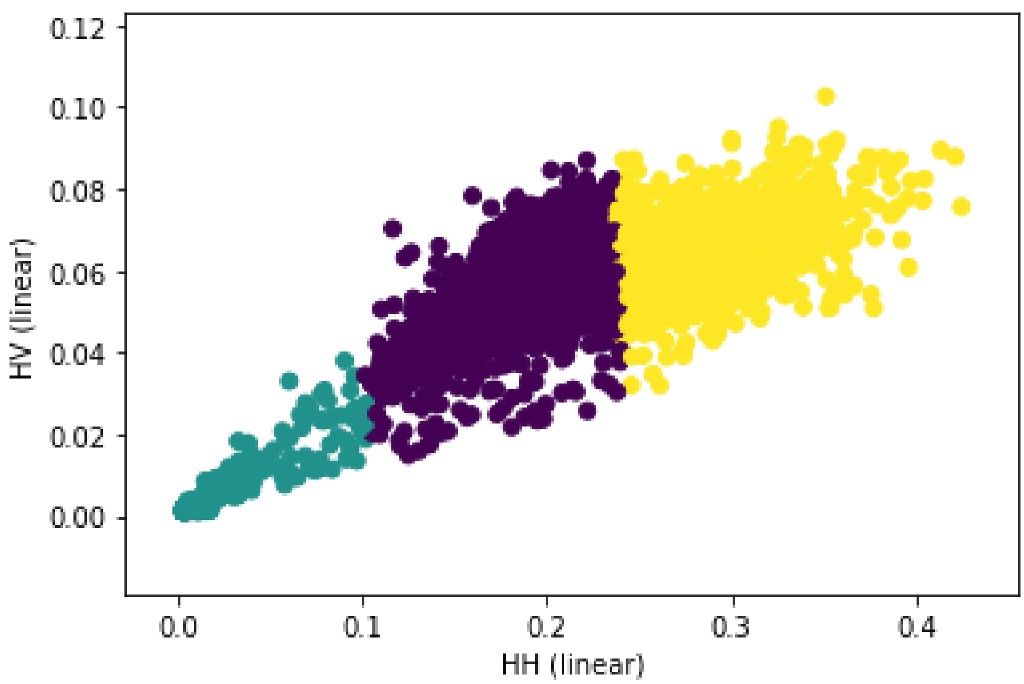
[21]:
```
<matplotlib.image.AxesImage at 0x7fcdb8014550>
```



[22]:
```python
## Answer
profile['dtype'] = 'int32'
profile['nodata'] = None
with rasterio.open('classes.tif', 'w', **profile) as ds:
    ds.write(classes, 1)
```

[23]:
```
## Answer

N = 4000
indices = np.random.choice(np.arange(hh_tv.size), N)
plt.scatter(hh_tv.ravel()[indices], hv_tv.ravel()[indices], c=classes.
 ↪ravel()[indices])
plt.xlabel('HH (linear)')
plt.ylabel('HV (linear)')
```

[23]: Text(0, 0.5, 'HV (linear)')



7 Acknowledgements

A portion of this research was performed at the Jet Propulsion Laboratory, California Institute of Technology. Copyright 2019 California Institute of Technology. US Government Support Acknowledged.

[ ]:
```

```

5_RGB_Composites

December 5, 2019

[1]:
```
import sys
sys.path.append('../utils')
from nd_tools import scale_img

import rasterio
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from tqdm import tqdm
```

The goal of this notebook is to visualize dual-pole backscatter images (HH, HV) as an RGB.

1 Inspecting the data

[2]:
```
data_dir = Path('../data')
backscatter_tifs = sorted(list(data_dir.glob('./*backscatter*.tif')))
backscatter_tifs
```

[2]:
```
[PosixPath('../data/alos_backscatter_hh.tif'),
 PosixPath('../data/alos_backscatter_hv.tif')]
```

[3]:
```
data_dir = Path('../data')
tv_tifs = sorted(list(data_dir.glob('./*tv.tif')))
tv_tifs
```

[3]:
```
[PosixPath('../data/alos_hh_tv.tif'), PosixPath('../data/alos_hv_tv.tif')]
```

2 RGB Composites

We are going to create an RGB composite following this guide. Specifically,

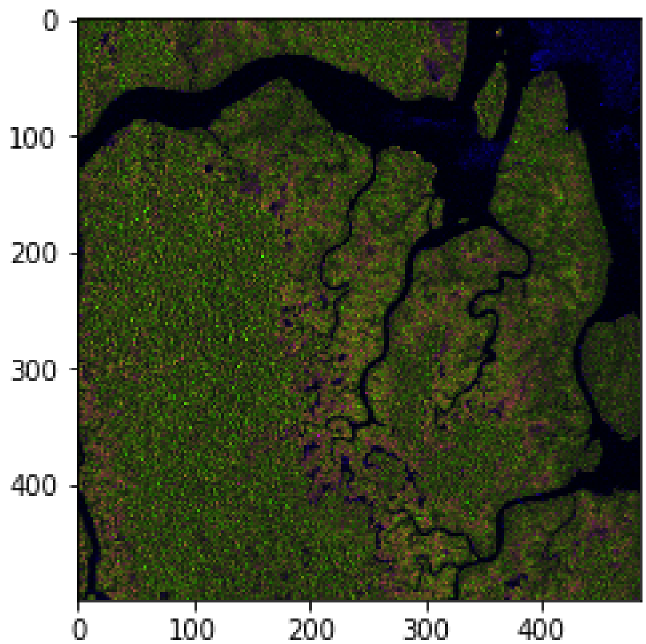
- Red = HH
- Green = HV
- blue = (HH / HV)

We scale each channel so that it lies between 0 and 1 so it can be read as in image by matplotlib.

[4]:
```
def open_one(path):
    with rasterio.open(path) as ds:
        band = ds.read(1)
    return band

bands = list(map(open_one, backscatter_tifs))
eps = .00001
bands += [bands[0] / (bands[1] + eps)]
```

[5]:
```
bands = list(map(scale_img, bands))
rgb = np.stack(bands, axis=2)
plt.imshow(rgb)
```

[5]: <matplotlib.image.AxesImage at 0x7fa0385f8d68>



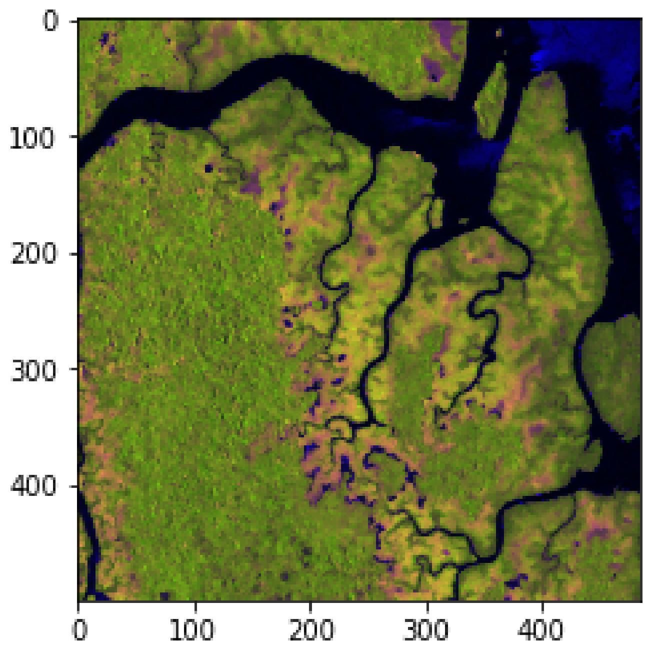
2.1 Excercise 1 - Make a TV Denoised RGB

Use the TV Denoised Images to create an RGB image as before.

[6]:
```
## Answer

bands_tv = list(map(open_one, tv_tifs))
eps = .00001
bands_tv += [bands_tv[0] / (bands_tv[1] + eps)]
```

[7]:
```
## Answer

temp = list(map(scale_img, bands_tv))
rgb_tv = np.stack(temp, axis=2)
plt.imshow(rgb_tv)
```

[7]: <matplotlib.image.AxesImage at 0x7fa028041978>



[8]:
```
## Answer

with rasterio.open(tv_tifs[0]) as ds:
    profile = ds.profile
profile['count'] = 3
with rasterio.open(data_dir/'rgb.tif', 'w', **profile) as ds:
    ds.write(np.moveaxis(rgb_tv, 2, 0))
```

3 Exercise 2 - Make an RGB from dB images

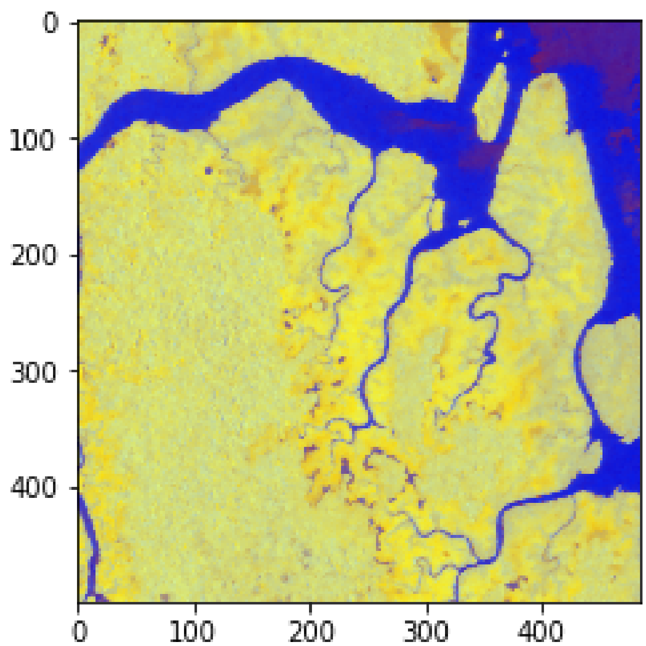
How does k-means run on this?

[9]:
```
## Answer

def open_one_db(path):
    with rasterio.open(path) as ds:
        band = 10 * np.log10(ds.read(1))
```

```
    return band


bands_db = list(map(open_one_db, tv_tifs))
eps = .00001
bands_db += [bands_db[0] / (bands_db[1] + eps)]
```

[10]:
```
## Answer

temp = list(map(scale_img, bands_db))
rgb_tv_db = np.stack(temp, axis=2)
plt.imshow(rgb_tv_db)
```

[10]: <matplotlib.image.AxesImage at 0x7fa0685ed390>



4 Exercise 3

Use the RGB from the previous exercise to perform k-means with three classes. What do you notice?

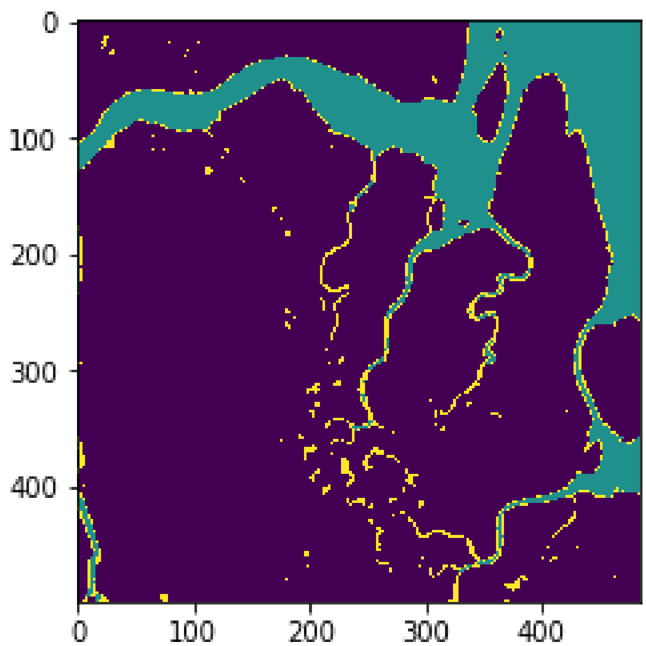
[11]:
```
## Answer
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3,
               random_state=0)
```

```
X_train = np.zeros((rgb_tv[..., 0].size, 3))
for k in range(2):
    X_train[:, k] = bands_db[k].ravel()
model.fit(X_train)
```

[11]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[12]:
```
## Answer

classes = model.labels_.reshape(bands_db[0].shape)
plt.imshow(classes)
```

[12]:
```
<matplotlib.image.AxesImage at 0x7fa08a617860>
```



5 Acknowledgements

We gratefully and humbly acknowledge JAXA for the use of ALOS-2 data to generate a coherence image over Mondah, Gabon generated using ISCE2.

We are grateful to the ASF for providing easy access to ALOS-1 radiometrically and terrain corrected tiles over Mondah and Ko Panyi.

A portion of this research was performed at the Jet Propulsion Laboratory, California Institute

6_Coherence

December 5, 2019

[1]:
```
import sys
sys.path.append('../utils')
from nd_tools import scale_img

import rasterio
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from tqdm import tqdm
from sklearn.cluster import KMeans
```

The goals of this notebook is to:

1. Introduce coherence images
2. Use coherence images for classification

1 Inspecting Data

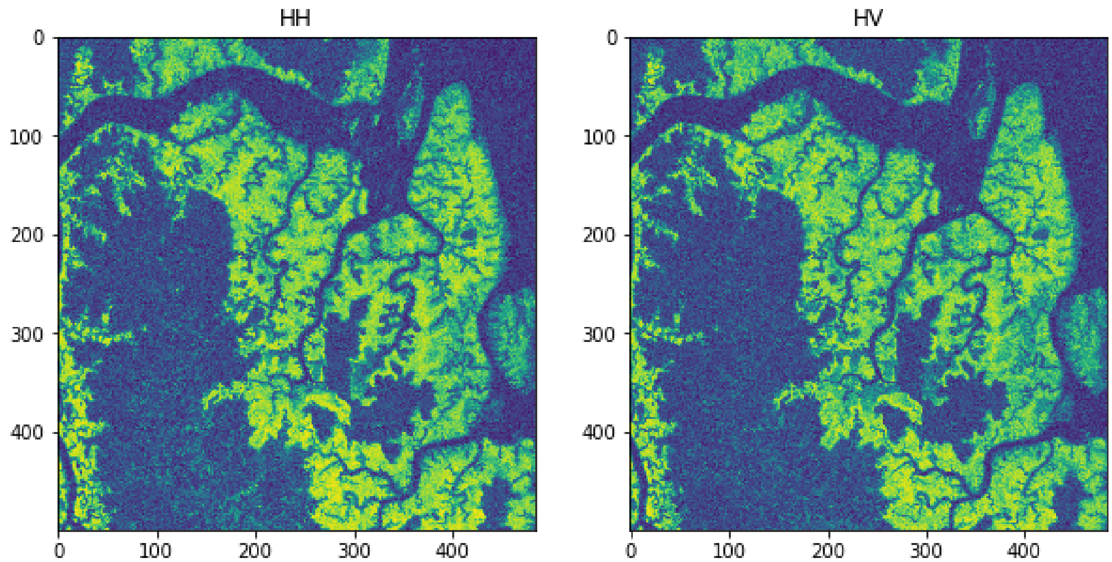
[2]:
```
data_dir = Path('../data')
coh_tifs = sorted(list(data_dir.glob('./*coh*.tif')))
coh_tifs
```

[2]:
```
[PosixPath('../data/alos_coh_hh.tif'), PosixPath('../data/alos_coh_hv.tif')]
```

[3]:
```
def open_one(path):
    with rasterio.open(path) as ds:
        band = ds.read(1)
    return band

coherence_bands = list(map(open_one, coh_tifs))
```

[4]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(coherence_bands[0])
ax[0].set_title('HH')
im = ax[1].imshow(coherence_bands[1])
ax[1].set_title('HV')
```

[4]: Text(0.5, 1.0, 'HV')

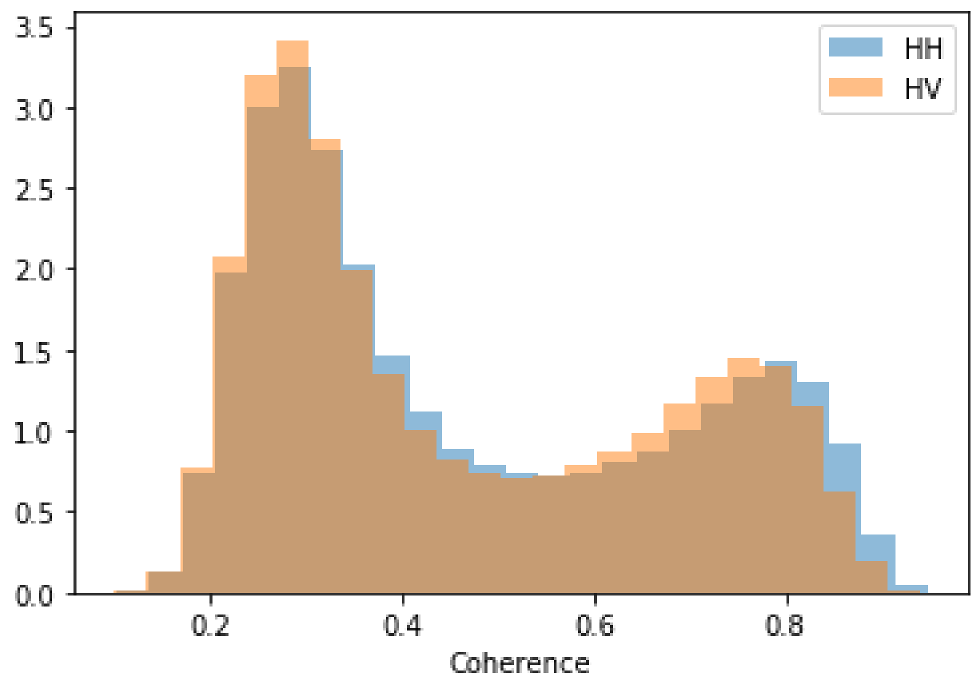


Coherence is a measure of complex correlation within a given window. Above, the coherence images are from ALOS-2 and processed using the open-source ISCE2. A nice tutorial on how to obtain coherence images from ALOS with ISCE can be found here.

We see indeed the correlation lies between 0 and 1.

[5]:
```
plt.hist(coherence_bands[0].ravel(),
         bins=25,
         density=True,
         label='HH',
         alpha=.5)
plt.hist(coherence_bands[1].ravel(),
         bins=25,
         density=True,
         label='HV',
         alpha=.5)
plt.xlabel('Coherence')
plt.legend()
```

[5]: <matplotlib.legend.Legend at 0x7ffc20021a90>



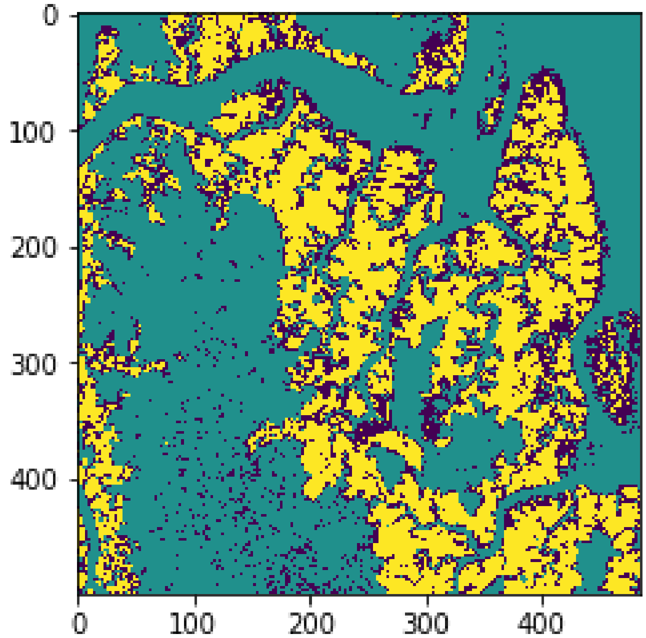
Let's see what happens when we classify the coherence image with k-means.

[6]:
```
model = KMeans(n_clusters=3,
               random_state=0)
X_train = np.zeros((coherence_bands[0].size, 2))
X_train[:, 0] = coherence_bands[0].ravel()
X_train[:, 1] = coherence_bands[1].ravel()
model.fit(X_train)
```

[6]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[7]:
```
classes = model.labels_.reshape(coherence_bands[0].shape)
plt.imshow(classes)
```

[7]:
```
<matplotlib.image.AxesImage at 0x7ffc1801ceb8>
```



Clearly the forest and ocean are conflated. Let's add backscatter to the classification.

2 Combining Coherence and Backscatter for Classification

[8]:
```
data_dir = Path('../data')
tv_tifs = sorted(list(data_dir.glob('./*tv.tif')))
tv_tifs
```

[8]:
```
[PosixPath('../data/alos_hh_tv.tif'), PosixPath('../data/alos_hv_tv.tif')]
```

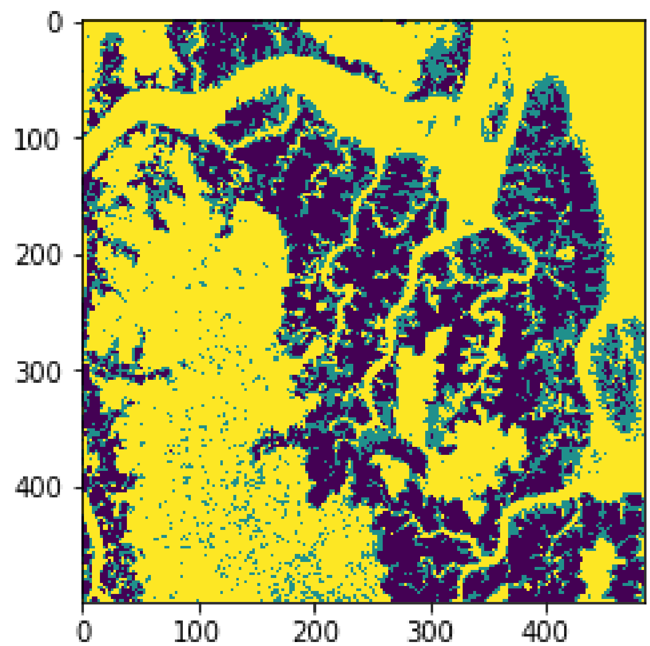
[9]:
```
backscatter_bands = list(map(open_one, tv_tifs))
```

[10]:
```
model = KMeans(n_clusters=3,
               random_state=0)
X_train = np.zeros((coherence_bands[0].size, 4))
X_train[:, 0] = coherence_bands[0].ravel()
X_train[:, 1] = coherence_bands[1].ravel()
X_train[:, 2] = backscatter_bands[0].ravel()
X_train[:, 3] = backscatter_bands[1].ravel()
model.fit(X_train)
```

[10]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[11]:
```
classes = model.labels_.reshape(backscatter_bands[0].shape)
plt.imshow(classes)
```

[11]: <matplotlib.image.AxesImage at 0x7ffc601dd438>



3 Excercise 1

Improve the classification by scaling each band to be between 0 and 1. Why does this help? Where are the problem points? Try using 4 classes, too.

Save to GTiff and inspect.

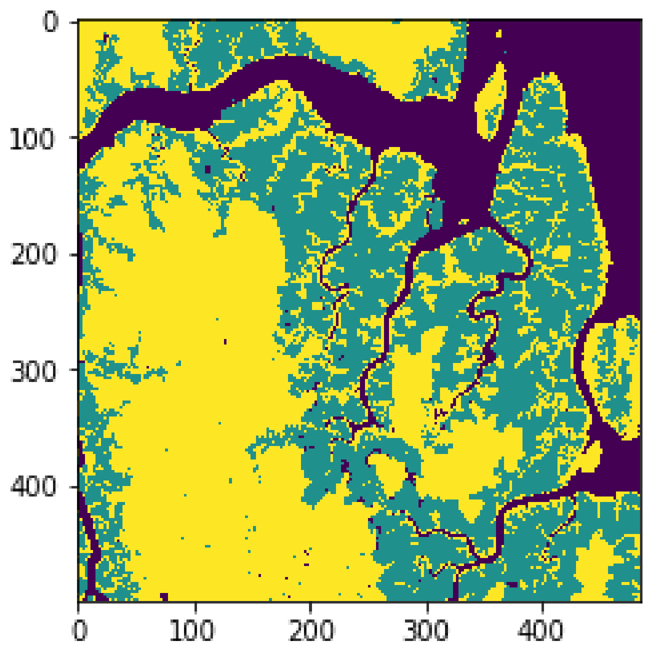
[12]:
```
## Answer

all_bands = backscatter_bands + coherence_bands
all_bands = list(map(scale_img, all_bands))
```

[13]:
```
## Answer

model = KMeans(n_clusters=3,
               random_state=0)
for k in range(len(all_bands)):
    X_train[:, k] = all_bands[k].ravel()
model.fit(X_train)
```

[13]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)

[14]:
```
## Answer

classes = model.labels_.reshape(backscatter_bands[0].shape)
plt.imshow(classes)
```

[14]: <matplotlib.image.AxesImage at 0x7ffc200a4fd0>



[15]:
```
## Answer

with rasterio.open(tv_tifs[0]) as ds:
    profile = ds.profile

profile['dtype'] = 'int32'
profile['nodata'] = None
with rasterio.open('classes.tif', 'w', **profile) as ds:
    ds.write(classes, 1)
```

4 Acknowledgements

We gratefully and humbly acknowledge JAXA for the use of ALOS-2 data to generate a coherence image over Mondah, Gabon generated using ISCE2.

We are grateful to the ASF for providing easy access to ALOS-1 radiometrically and terrain corrected tiles over Mondah and Ko Panyi.

A portion of this research was performed at the Jet Propulsion Laboratory, California Institute of Technology. US Government Support Acknowledged.

7_Superpixel_Segmentation

December 5, 2019

[4]:
```
import sys
sys.path.append('../utils')
from nd_tools import (scale_img,
                      get_superpixel_means_as_features,
                      get_superpixel_stds_as_features,
                      get_array_from_features)

from rio_tools import polygonize_array_to_shapefile
import rasterio
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from tqdm import tqdm
from sklearn.cluster import KMeans
from skimage.segmentation import felzenszwalb
from skimage.color import label2rgb
```

The goals of this notebook are to:

1. Introduce superpixel segmentation as a tool for feature extraction
2. How to use superpixels with SAR imagery and classification

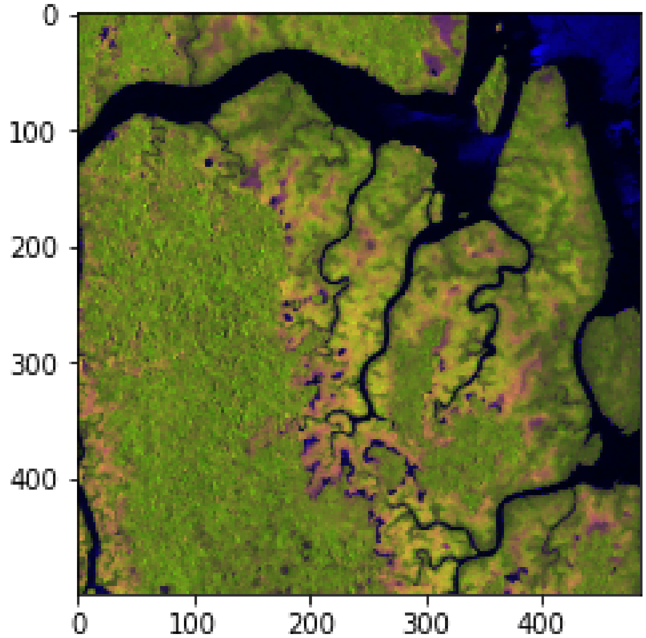
1 Inspecting Data

[5]:
```
data_dir = Path('../data')
rgb_tif = data_dir/'rgb.tif'
coh_tifs = sorted(list(data_dir.glob('./*coh*.tif')))
coh_tifs
```

[5]:
```
[PosixPath('../data/alos_coh_hh.tif'), PosixPath('../data/alos_coh_hv.tif')]
```

[6]:
```
with rasterio.open(rgb_tif) as ds:
    rgb = ds.read().transpose([1, 2, 0])
    profile = ds.profile
plt.imshow(rgb)
```

[6]:
```
<matplotlib.image.AxesImage at 0x7fdc28543898>
```



2 Superpixel Segmentation

The notion of the "superpixel" were first introduced in this article as a way segmenting in an image into small homogeneous areas. Frequently, individual pixel analysis obscures higher level image analysis such as object detection while lowerin the resolution performs unwanted smoothing. Since superpixels were introduced, there have been numerous subsequent algorithms for extracting superpixel segmentations including the Felzenszwalb algorithm which is implemented in skimage and nicely demonstrated here. This is the algorithm we use.

For remote sensing, superpixels mitigate noise within a small homogenous image areas particularly for classifcation 1 2. Additionally, they are used in global scale Mangrove extent 3 and forest loss 4.

2.1 Using Skimage

Skimage takes a multichannel image (in this case our RGB composite) and outputs a matrix of labels so that each pixel has a unique integer label.

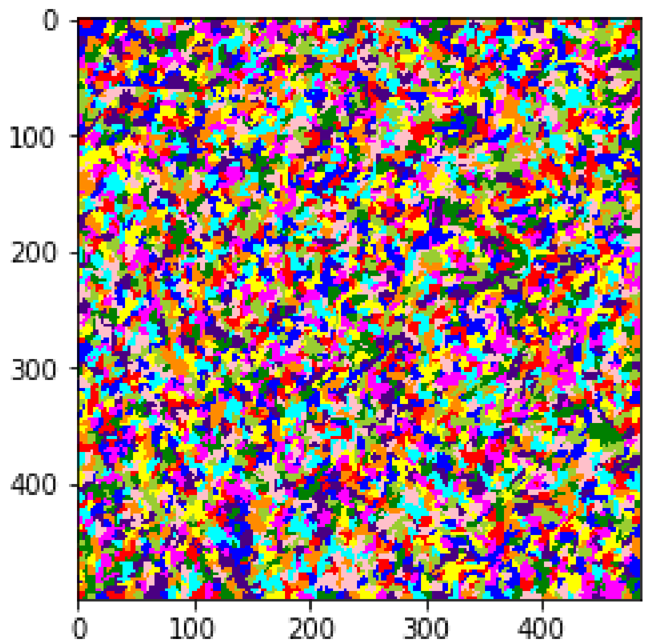
[7]:
```
# Check the documentation and papers above for an explanation of parameters
# We don't use sigma because TV is already denoised
superpixel_labels = felzenszwalb(rgb, scale=.5, sigma=0, min_size=15,␣
 ↪multichannel=True)
print('superpixels shape: ', superpixel_labels.shape)
print('rgb shape: ', rgb.shape)
```

```
superpixels shape:  (500, 484)
rgb shape:  (500, 484, 3)
```

We now visualize the array with each segment as a random color.

[8]:
```python
superpixel_labels_viz = label2rgb(superpixel_labels)
plt.imshow(superpixel_labels_viz)
```

[8]:
```
<matplotlib.image.AxesImage at 0x7fdc5850aa58>
```



3 Viewing Segments in QGIS

To view in QGIS, we use the `features` module from QGIS and `fiona` to write these vectors to file. We have collected this answer from GIS stackexchange into a function.

[9]:
```python
polygonize_array_to_shapefile(superpixel_labels, profile, 'superpixels')
```

4 Extracting Features from Segments

Within a segment we can extract means, variance, medians, etc. We use the measurement module from scipy's ndimage. We have some functions to make their use slightly more straightforward.

[10]:
```python
std_features = get_superpixel_stds_as_features(superpixel_labels, rgb)
mean_features = get_superpixel_means_as_features(superpixel_labels, rgb)
```

```
/Users/cmarshak/anaconda3/envs/sari_test/lib/python3.7/site-
packages/scipy/ndimage/measurements.py:523: RuntimeWarning: invalid value
encountered in true_divide
  means = sums / counts
```

We have features $K \times q$ where K are the number of superpixels segments and q are the number of channels. Skimage also ensures that are labels are of the form 0, 1, 2, … K, so that each index of the mean feature is also its label, which makes subsequent manipulation a bit easier.

[11]:
```python
print('mean features shape: ', mean_features.shape)
print('Number of labels: ', len(np.unique(superpixel_labels)))
```

```
mean features shape:  (4772, 3)
Number of labels:  4772
```

To retiterate, (K unique labels) $\times$ (q channels). Here $q = 3$ as we have an rgb image with three channels.

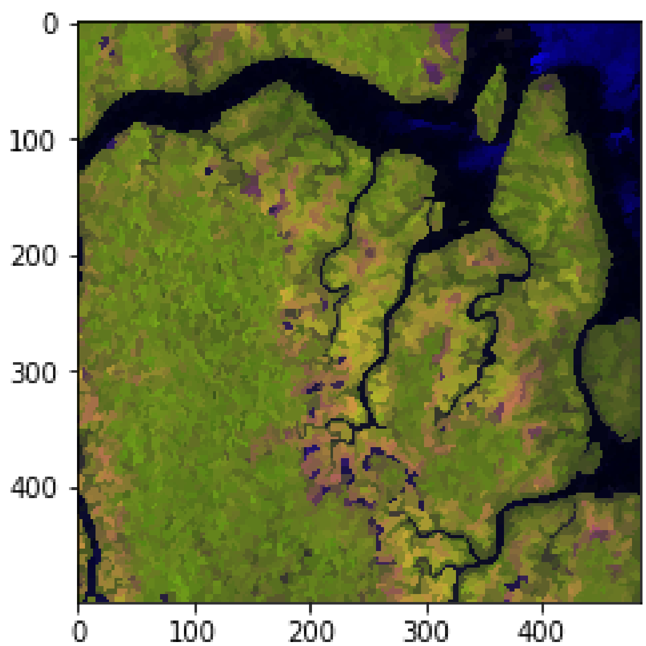
5 Mean Populated Superpixels

We want to create an array corresponding to our map, where every pixel of a particular label is populated with the mean of the superpixel it belongs to. This is accomplished using `get_array_from_features`, which again is a wrapper for `labeled_comprehension` from scipy.

[12]:
```python
mean_arr = get_array_from_features(superpixel_labels, mean_features)
```

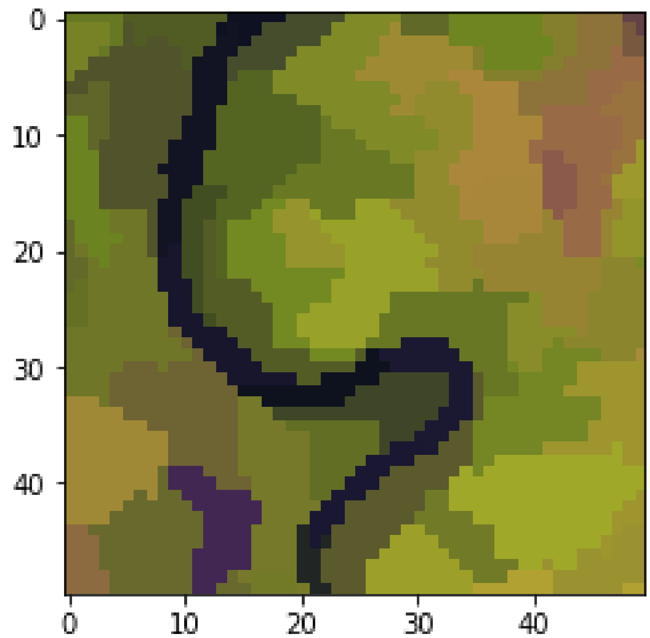
[13]:
```python
plt.imshow(mean_arr)
```

[13]: <matplotlib.image.AxesImage at 0x7fdc3917b828>



[14]:
```
plt.imshow(mean_arr[200:250, 200:250])
```

[14]: <matplotlib.image.AxesImage at 0x7fdc390d6240>



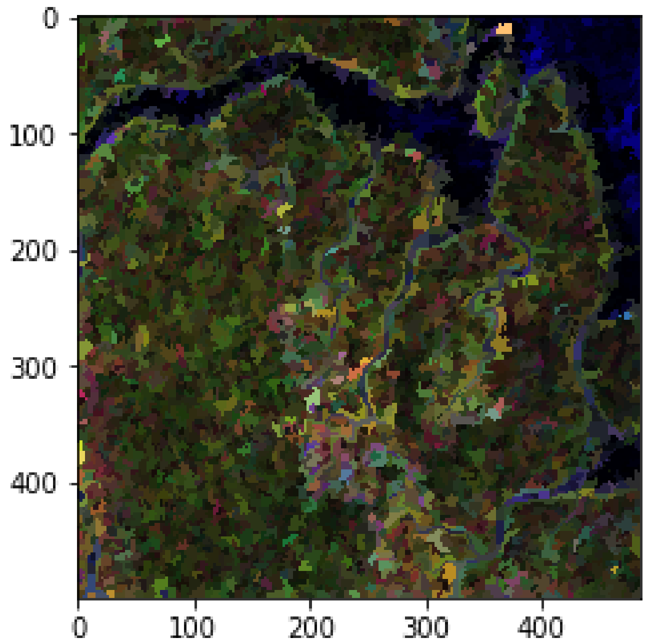
5.1 Standard Deviation Populated Superpixels

Using standard deviation features, we can get a proxy for texture or how quickly the pixel values vary in a superpixel segment. Although we do not use this features for this tutorial, we mention it as a useful tool for your future analyses.

[15]:
```
std_arr = get_array_from_features(superpixel_labels, std_features)
# because the variance is lower, without scaling the image can't be viewed in␣
↪imshow - scales to 0-1
std_arr = scale_img(std_arr)
```

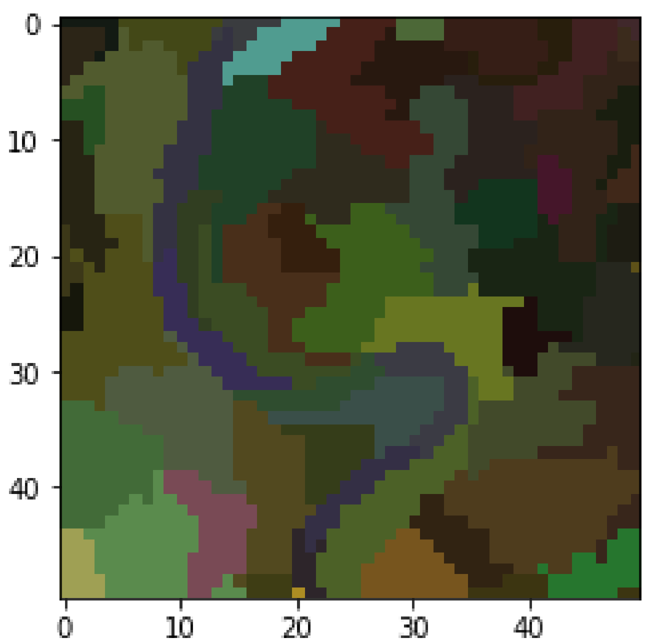
[16]:
```
plt.imshow(std_arr)
```

[16]: <matplotlib.image.AxesImage at 0x7fdbd810d7b8>



[17]:
```
plt.imshow(std_arr[200:250, 200:250])
```

[17]: <matplotlib.image.AxesImage at 0x7fdc39158f60>



6 K-means with Superpixels

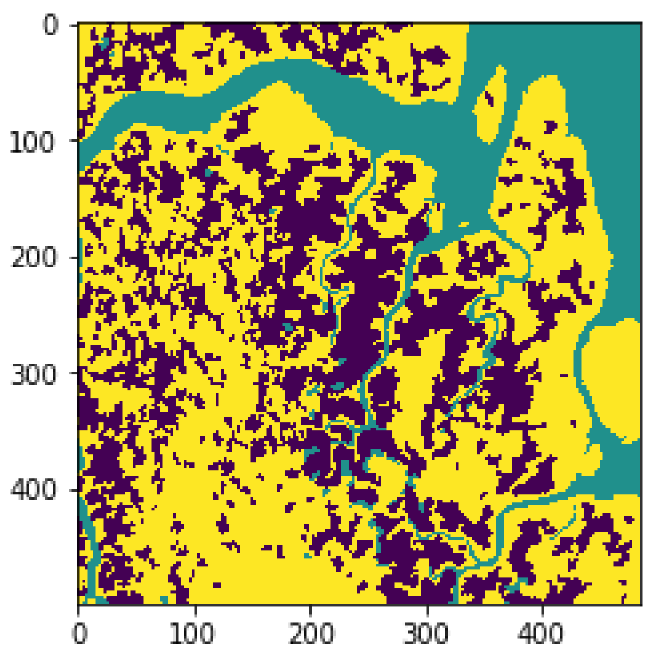
Superpixels can improve model performance by grouping homogeneous areas. We now show how to apply sklearn models to the superpixel features. We use the mean features.

[18]:
```
## Answer

model = KMeans(n_clusters=3,
               random_state=0)
X_train = mean_features.copy()
model.fit(X_train)
```

[18]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[19]:
```
## Answer

class_features = model.labels_.reshape((-1, 1))
classes = get_array_from_features(superpixel_labels, class_features)
plt.imshow(classes)
```

[19]:
```
<matplotlib.image.AxesImage at 0x7fdc395a9390>
```



7 Exercise 1

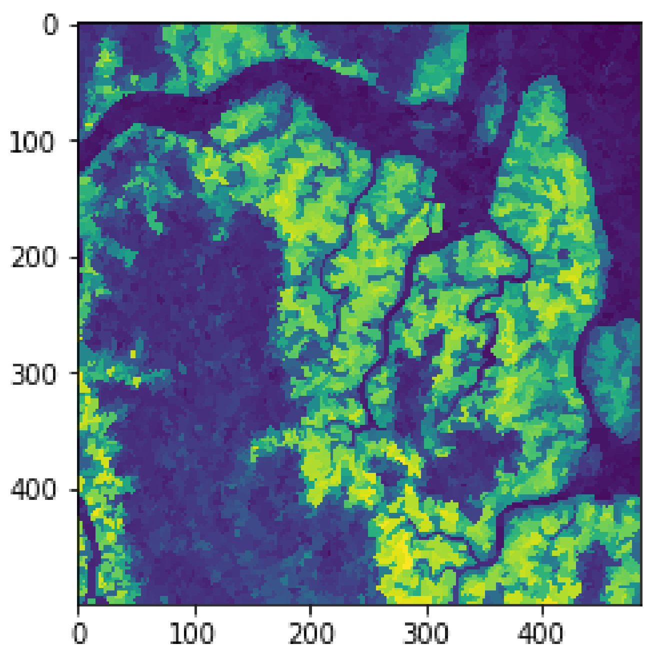
Use the superpixels found above to aggregate HH coherence using the mean.

[20]:
```
## Answer

with rasterio.open(coh_tifs[0]) as ds:
    coh_hh = ds.read(1)
```

[21]:
```
## Answer
coh_mean_features = get_superpixel_means_as_features(superpixel_labels, coh_hh)
```

[22]:
```
## Answer
coh_mean_arr = get_array_from_features(superpixel_labels, coh_mean_features)
plt.imshow(coh_mean_arr)
```

[22]: `<matplotlib.image.AxesImage at 0x7fdbd80cb9e8>`



8 Exercise 2

Use the superpixels mean coherence to classify the image with k-means. What do you notice?

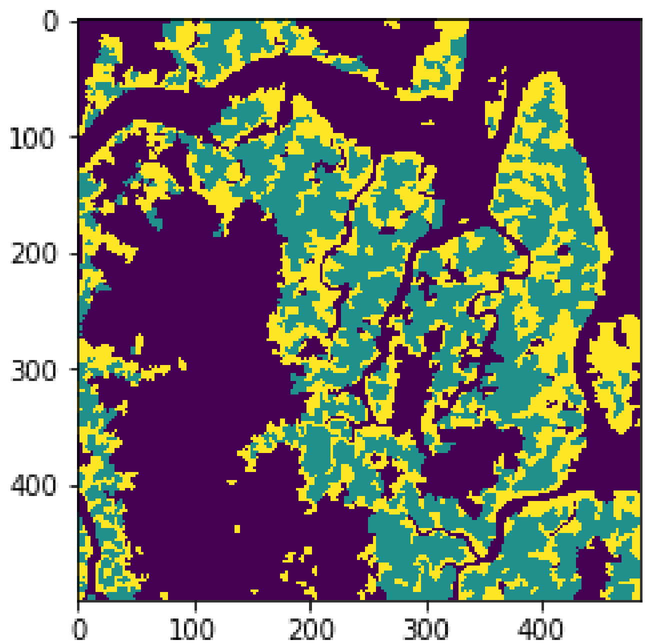
[23]:
```
## Answer

model = KMeans(n_clusters=3,
```

```
                 random_state=0)
X_train = coh_mean_features.copy()
model.fit(X_train)
```

[23]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
        n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
        random_state=0, tol=0.0001, verbose=0)
```

[24]:
```
## Answer

class_features = model.labels_.reshape((-1, 1))
classes = get_array_from_features(superpixel_labels, class_features)
plt.imshow(classes)
```

[24]:
```
<matplotlib.image.AxesImage at 0x7fdbd80db358>
```



That's a pretty good classification, but what two classes are being conflated? Ideas for separating them?

9 Exercise 3

Combine coherence and backscatter means from the superpixels to obtain a classification.

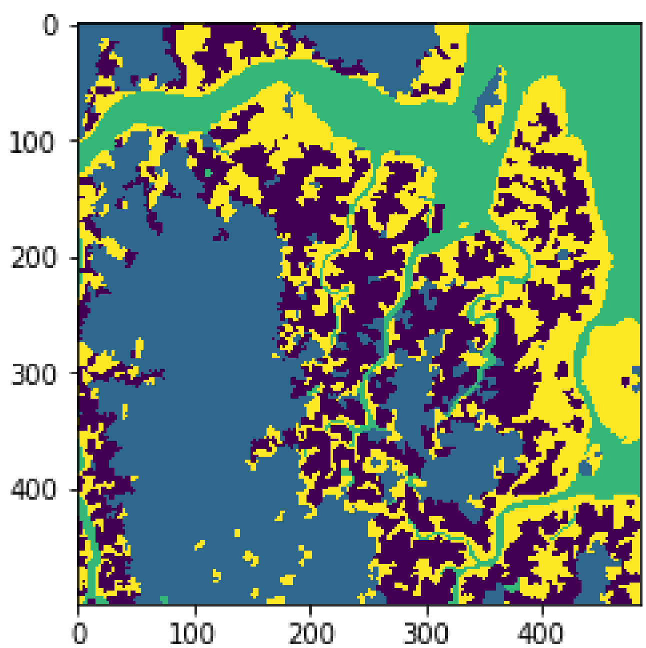
[25]:
```
## Answer
```

```
model = KMeans(n_clusters=4,
               random_state=0)
X_train = np.concatenate([mean_features, coh_mean_features], axis=1)
model.fit(X_train)
```

[25]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[26]:
```
## Answer

class_features = model.labels_.reshape((-1, 1))
classes = get_array_from_features(superpixel_labels, class_features)
plt.imshow(classes)
```

[26]:
```
<matplotlib.image.AxesImage at 0x7fdc28851a20>
```



[27]:
```
## Answer

with rasterio.open(coh_tifs[0]) as ds:
    profile = ds.profile

profile['dtype'] = 'int32'
profile['nodata'] = None
with rasterio.open('classes.tif', 'w', **profile) as ds:
```

```
    ds.write(classes.astype(np.int32), 1)
```

10 Acknowledgements

We gratefully and humbly acknowledge JAXA for the use of ALOS-2 data to generate a coherence image over Mondah, Gabon generated using ISCE2.

We are grateful to the ASF for providing easy access to ALOS-1 radiometrically and terrain corrected tiles over Mondah and Ko Panyi.

A portion of this research was performed at the Jet Propulsion Laboratory, California Institute of Technology. US Government Support Acknowledged.

8_Change_Detection

December 5, 2019

[1]:
```
import sys
sys.path.append('../utils')
from nd_tools import (scale_img,
                      get_superpixel_means_as_features,
                      get_array_from_features,
                      filter_binary_array_by_min_size)

from rio_tools import polygonize_array_to_shapefile
import rasterio
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
from sklearn.cluster import KMeans
from skimage.segmentation import felzenszwalb
from skimage.color import label2rgb
from skimage.restoration import denoise_tv_bregman
import pandas
import pandas as pd
import seaborn as sns
```

The goal of this notebook is to demonstrate how to utilize the same open source tools discussed in the previous notebooks to detect forest loss. Here, we identify forest loss according to decrease in HV backscatter as done globally here - JAXA uses proprietary eCognition software for their segmentation but we will use skimage, which is open source.

We downloaded the data from the ASF search tool over the Thai city Ko Panyi. The link to the original ALOS-1 data tiles can be downloaded here: first image and second image (you will have to be signed in Earth Data to download). We selected these two images over the geographic area so that:

- The images are RTC-ed - no dependence on incidence angle.
- The images are low resolution - permits quick exploratory analysis - high resolution images are available, but require more disk space and more processing time.
- The images are acquired during summer months for fair year-to-year comparison and when there is little rain which can impact backscatter.
- The images are dual polarization (HH and HV) so we can construct the RGB as done previously.
- There are nodata/nan areas when you download the data. For this tutorial, we fill nodata areas using inpainting from example from skimage.

This tutorial demonstrates how to download data from a site of your choosing (click event recording and give permission to download the application Adobe Connect). The raw data above was cropped and acquisitions dates were appended to the file name.

1 Inspecting the data

[2]:
```
data_dir = Path('data')
tifs = sorted(list(data_dir.glob('*.tif')))
tifs
```

[2]:
```
[PosixPath('data/AP_07915_FBD_F0150_RT2_HH_20070720.tif'),
 PosixPath('data/AP_07915_FBD_F0150_RT2_HV_20070720.tif'),
 PosixPath('data/AP_12612_FBD_F0150_RT2_HH_20080606.tif'),
 PosixPath('data/AP_12612_FBD_F0150_RT2_HV_20080606.tif')]
```

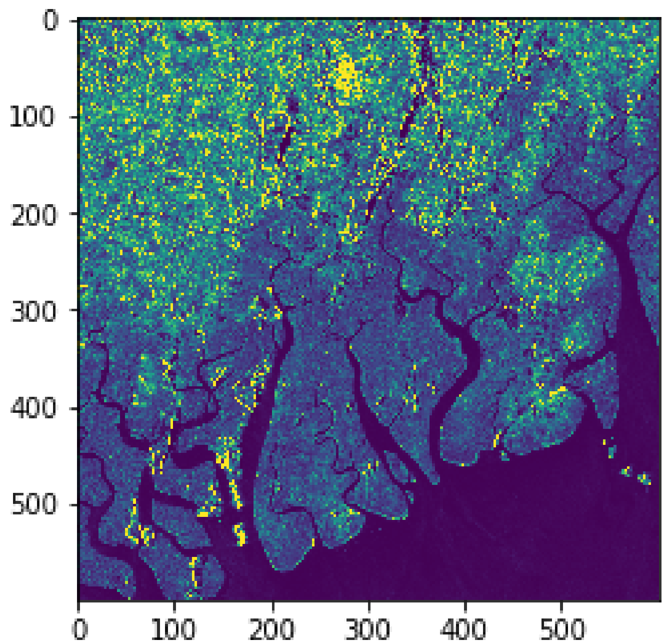
[3]:
```
def open_one(path):
    with rasterio.open(path) as ds:
        band = ds.read(1)
    return band

bands = list(map(open_one, tifs))

with rasterio.open(tifs[0]) as ds:
    profile = ds.profile
```

[4]:
```
profile
```

[4]:
```
{'driver': 'GTiff', 'dtype': 'float32', 'nodata': nan, 'width': 600, 'height':
600, 'count': 1, 'crs': CRS.from_epsg(32647), 'transform': Affine(30.0, 0.0,
440228.8125,
       0.0, -30.0, 937448.125), 'tiled': False, 'compress': 'packbits',
'interleave': 'band'}
```

[5]:
```
plt.imshow(bands[0])
```

[5]:
```
<matplotlib.image.AxesImage at 0x7fb2c017f6d8>
```



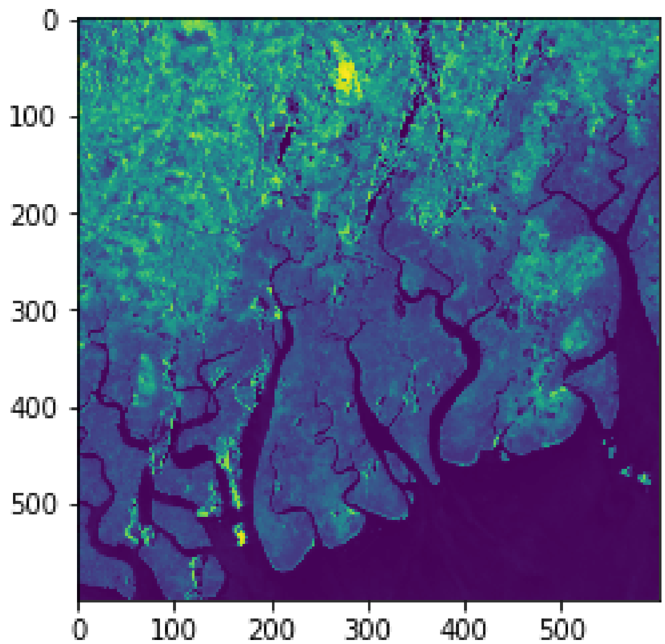
2 Despeckle

[6]:
```
def fwd(img):
    return 10 * np.log10(img)

def bwd(img):
    return 10**(img / 10)

def tv_denoise(img, weight):
    img_db = fwd(img)
    img_db_tv = denoise_tv_bregman(img_db, weight)
    img_tv = bwd(img_db_tv)
    return img_tv

def tv_partial(band):
    return tv_denoise(band, .5)
```

[7]:
```
bands_tv = list(map(tv_partial, bands))
```

[8]:
```
plt.imshow(bands_tv[0])
```

[8]: <matplotlib.image.AxesImage at 0x7fb30a78de48>



2.1 Write despeckles images

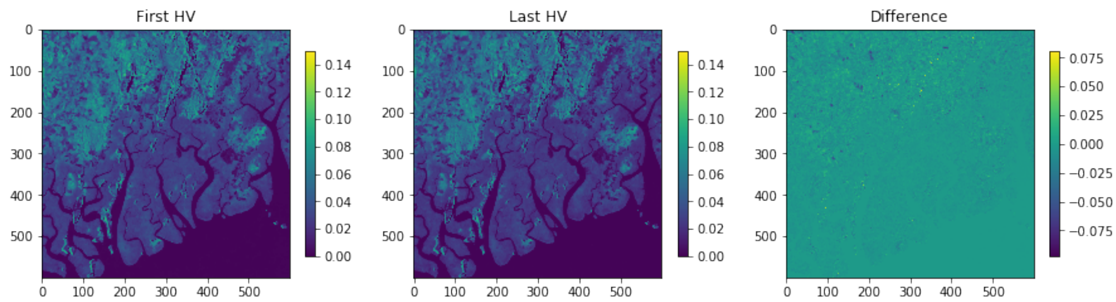
[9]:
```
hv_0 = bands_tv[1]
hv_1 = bands_tv[3]
hh_0 = bands_tv[0]
hh_1 = bands_tv[2]

diff_hv = hv_1 - hv_0
diff_hh = hh_1 - hh_0
```

[10]:
```
p = profile.copy()
p['count'] = 1
with rasterio.open('hv_0.tif', 'w', **p) as ds:
    ds.write(hv_0, 1)
with rasterio.open('hv_1.tif', 'w', **p) as ds:
    ds.write(hv_1, 1)

with rasterio.open('hh_0.tif', 'w', **p) as ds:
    ds.write(hh_0, 1)
with rasterio.open('hh_1.tif', 'w', **p) as ds:
    ds.write(hh_1, 1)
```

2.2 Visualize

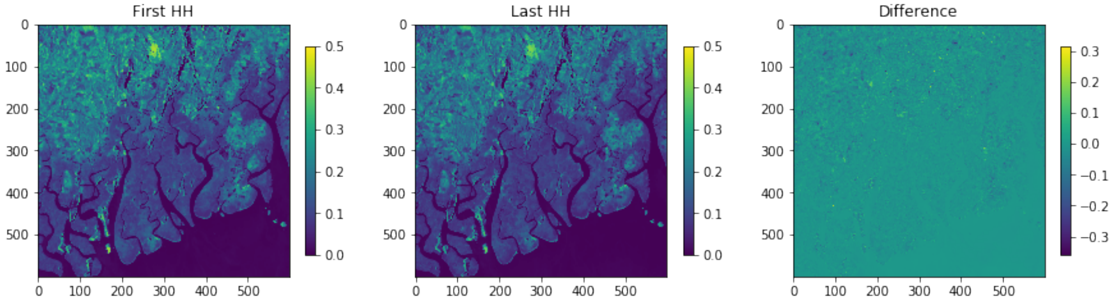
[11]:
```
fig, ax = plt.subplots(1, 3, figsize=(15, 10))

im = ax[0].imshow(hv_0, vmax=.15)
ax[0].set_title('First HV')
fig.colorbar(im, ax=ax[0], shrink=.3)


im = ax[1].imshow(hv_1, vmax=.15)
ax[1].set_title('Last HV')
fig.colorbar(im, ax=ax[1], shrink=.3)

ax[2].set_title('Difference')
im = ax[2].imshow(diff_hv)
fig.colorbar(im, ax=ax[2], shrink=.3)
```

[11]: <matplotlib.colorbar.Colorbar at 0x7fb30abe06d8>



[12]:
```
fig, ax = plt.subplots(1, 3, figsize=(15, 10))

im = ax[0].imshow(hh_0, vmax=.5)
ax[0].set_title('First HH')
fig.colorbar(im, ax=ax[0], shrink=.3)


im = ax[1].imshow(hh_1, vmax=.5)
ax[1].set_title('Last HH')
fig.colorbar(im, ax=ax[1], shrink=.3)

ax[2].set_title('Difference')
im = ax[2].imshow(diff_hh)
fig.colorbar(im, ax=ax[2], shrink=.3)
```

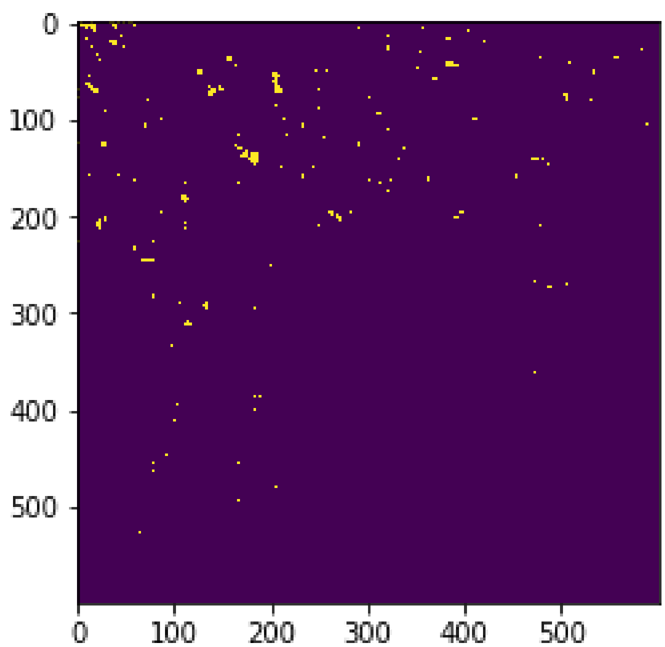
[12]: <matplotlib.colorbar.Colorbar at 0x7fb30acb3ef0>



2.3 Select a threshold for the HV Difference

[13]:
```python
change_class = (diff_hv < -.025).astype(int)
plt.imshow(change_class)
```

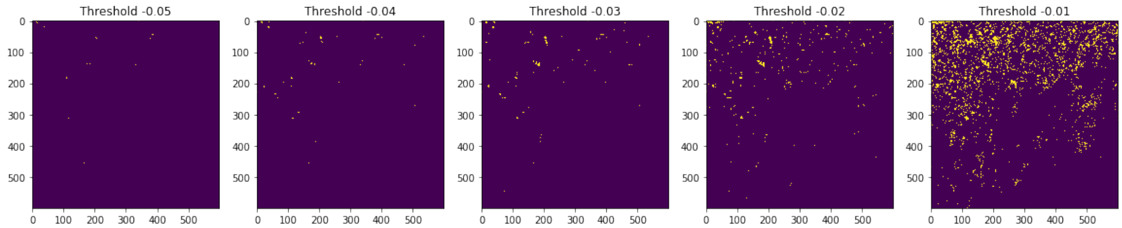
[13]:
```
<matplotlib.image.AxesImage at 0x7fb30b31e6d8>
```



We can experiment with a range of thresholds between -0.05 and -0.01.

[14]:
```python
thresholds = np.linspace(-.05, -.01,5)
N = len(thresholds)
fig, ax = plt.subplots(1, N, figsize=(20, 10))
for (ax, t) in zip(ax.ravel(), thresholds):
    ax.imshow((diff_hv < t).astype(int))
```

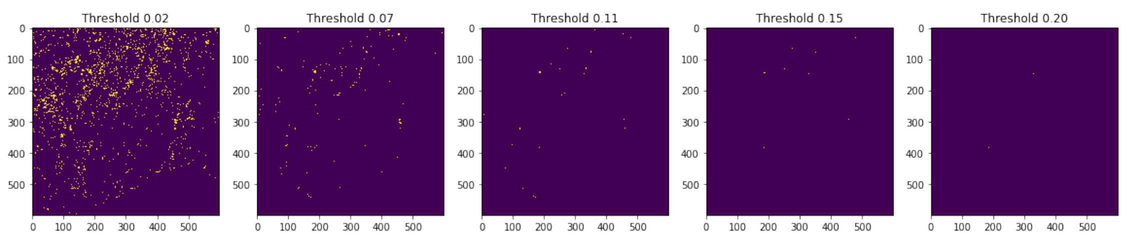
```
    ax.set_title(f'Threshold {t:1.2f}')
```



3 Exercise 1

Experiment with thresholds to determine areas where HH increases.

[15]:
```
## Answer

thresholds = np.linspace(.02, .2,5)
N = len(thresholds)
fig, ax = plt.subplots(1, N, figsize=(20, 10))
for (ax, t) in zip(ax.ravel(), thresholds):
    ax.imshow((diff_hh > t).astype(int))
    ax.set_title(f'Threshold {t:1.2f}')
```



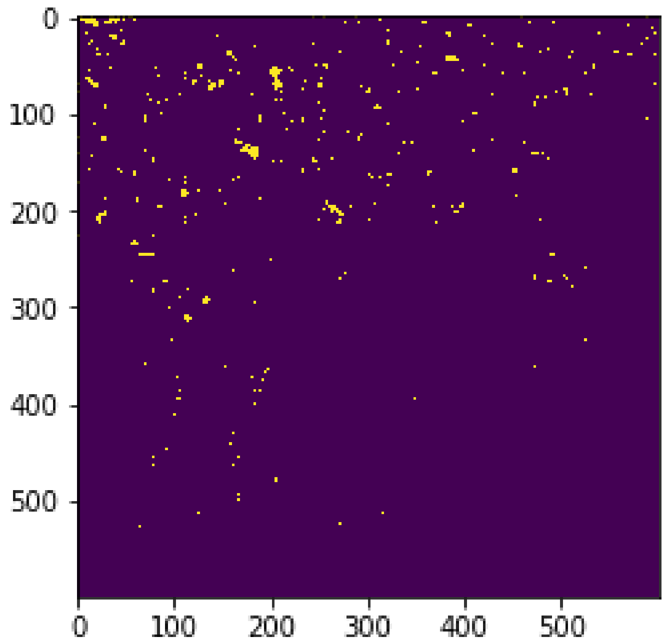
3.1 Morphological Filtering

We see there are lots of small areas where we detected change. This could be the residual noise in the image or just small environmental changes such as a truck traveling down a previously empty road.

We can filter an change areas by size. Specifically, we segment a binary array into contiguous parts using `label` and then measure each segments size. Using these labels and size, we only consider change areas of a specific size.

[16]:
```
change_class = (diff_hv < -.02).astype(int)
plt.imshow(change_class)
```

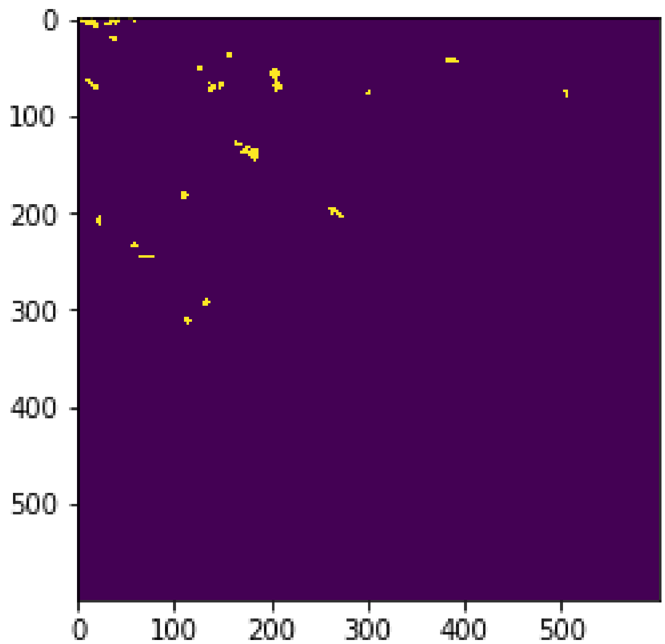
[16]: <matplotlib.image.AxesImage at 0x7fb2f843d748>



Below, we filter those areas that are less than 25 pixels.

[17]:
```
change_class_filtered = filter_binary_array_by_min_size(change_class, 25)

plt.imshow(change_class_filtered)
```

[17]: <matplotlib.image.AxesImage at 0x7fb30c811ba8>



[18]:
```
polygonize_array_to_shapefile(change_class_filtered,
                              profile,
                              'changes_hv_decrease',
                              mask=~(change_class_filtered.astype(bool)))
```

4 Using Superpixels

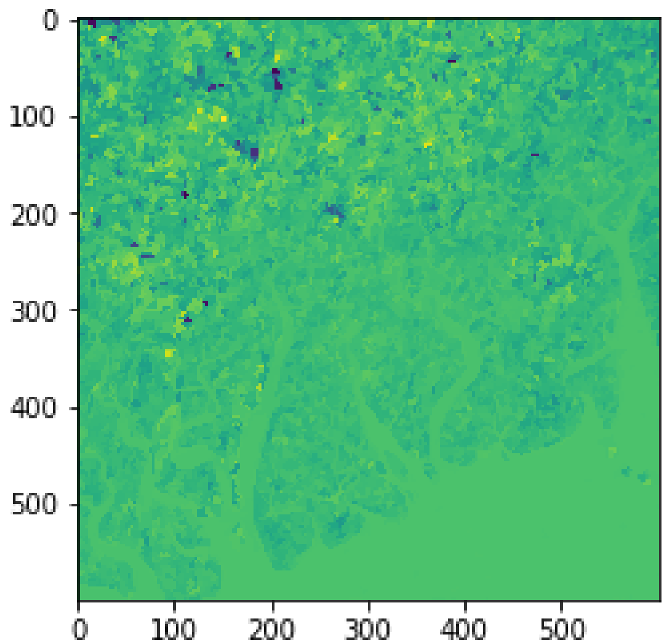
We may wish to use superpixels rather than morphological filtering to ensure large enough change areas are registered from the outset. In fact, superpixels can track areas through an entire time series of images as in our paper.

Let's segment according to the first and last HV image.

[19]:
```
superpixel_labels = felzenszwalb(np.stack([hv_0, hv_1], axis=2),
                                 scale=.5,
                                 sigma=0,
                                 min_size=15,
                                 multichannel=True)
```

[20]:
```
mean_features = get_superpixel_means_as_features(superpixel_labels, diff_hv)
mean_array = get_array_from_features(superpixel_labels, mean_features)
plt.imshow(mean_array)
```

[20]:
```
<matplotlib.image.AxesImage at 0x7fb30b86f9b0>
```



5 Exercise 2

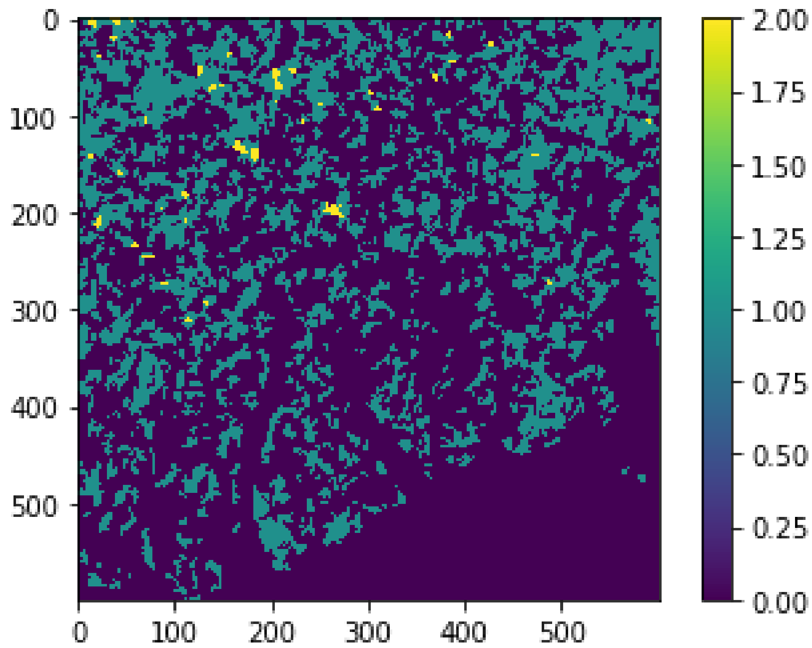
Use k-means to determine the change threshold using the superpixels found above and save to a shapefile. I used three classes.

[21]:
```
## Answer

model = KMeans(n_clusters=3,
               random_state=0,
           )
X_train = mean_features.copy()
model.fit(X_train)
```

[21]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

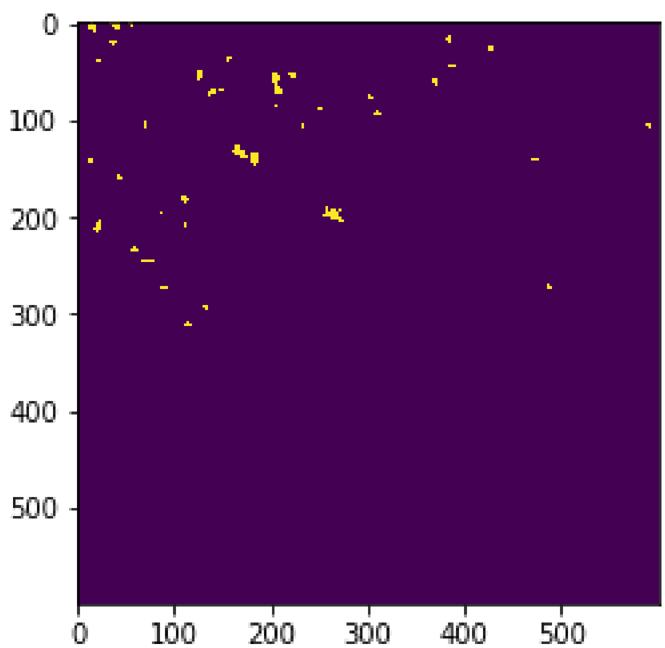
[22]:
```
## Answer

class_features = model.labels_.reshape((-1, 1))
classes = get_array_from_features(superpixel_labels, class_features)
plt.imshow(classes)
plt.colorbar()
```

[22]:
```
<matplotlib.colorbar.Colorbar at 0x7fb2a8a7b8d0>
```



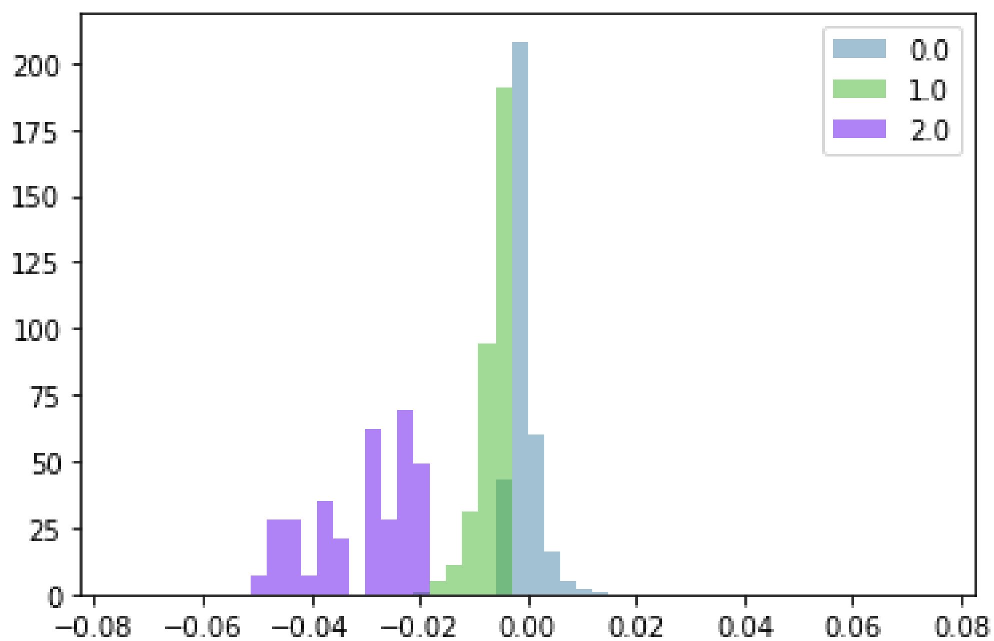
[23]:
```
## Answer


plt.imshow(classes == 2)
```

[23]: <matplotlib.image.AxesImage at 0x7fb300287668>



[24]:
```python
## Answer

for class_num in np.unique(classes):
    plt.hist(mean_features[class_features == class_num],
             range=(-.075, .075),
             bins=50,
             alpha=.5,
             label=str(class_num),
             density=True,
             color=[np.random.random(3)])
plt.legend()
```

[24]: <matplotlib.legend.Legend at 0x7fb2a8ecd978>



[25]:
```python
## Answer

changes = (classes == 2).astype(int)
polygonize_array_to_shapefile(changes,
                              profile,
                              'change_superpixel_hv_loss',
                              mask=~(changes.astype(bool)))
```

One could easily identify the class with greates hv loss to remove the dependence of the above approach on the user.

6 Analyzing Statistics Between Our Pair

We have the following statistics we can inspect by hand:

1. HV from the first image
2. HV from the last image
3. HH from the first image
4. HH from the last image

We can visualize the statistics of the above quanties with respect to our labeled areas. Maybe there are interesting relationships of scattering that we can more easily see with the pair plot.

To do this, we have to create a pandas dataframe, which you should think of as a spreadsheet stored in python.

[26]:
```
mean_features_hv_0 = get_superpixel_means_as_features(superpixel_labels, hv_0)
mean_features_hv_1 = get_superpixel_means_as_features(superpixel_labels, hv_1)
mean_features_hh_0 = get_superpixel_means_as_features(superpixel_labels, hh_0)
mean_features_hh_1 = get_superpixel_means_as_features(superpixel_labels, hh_1)
```

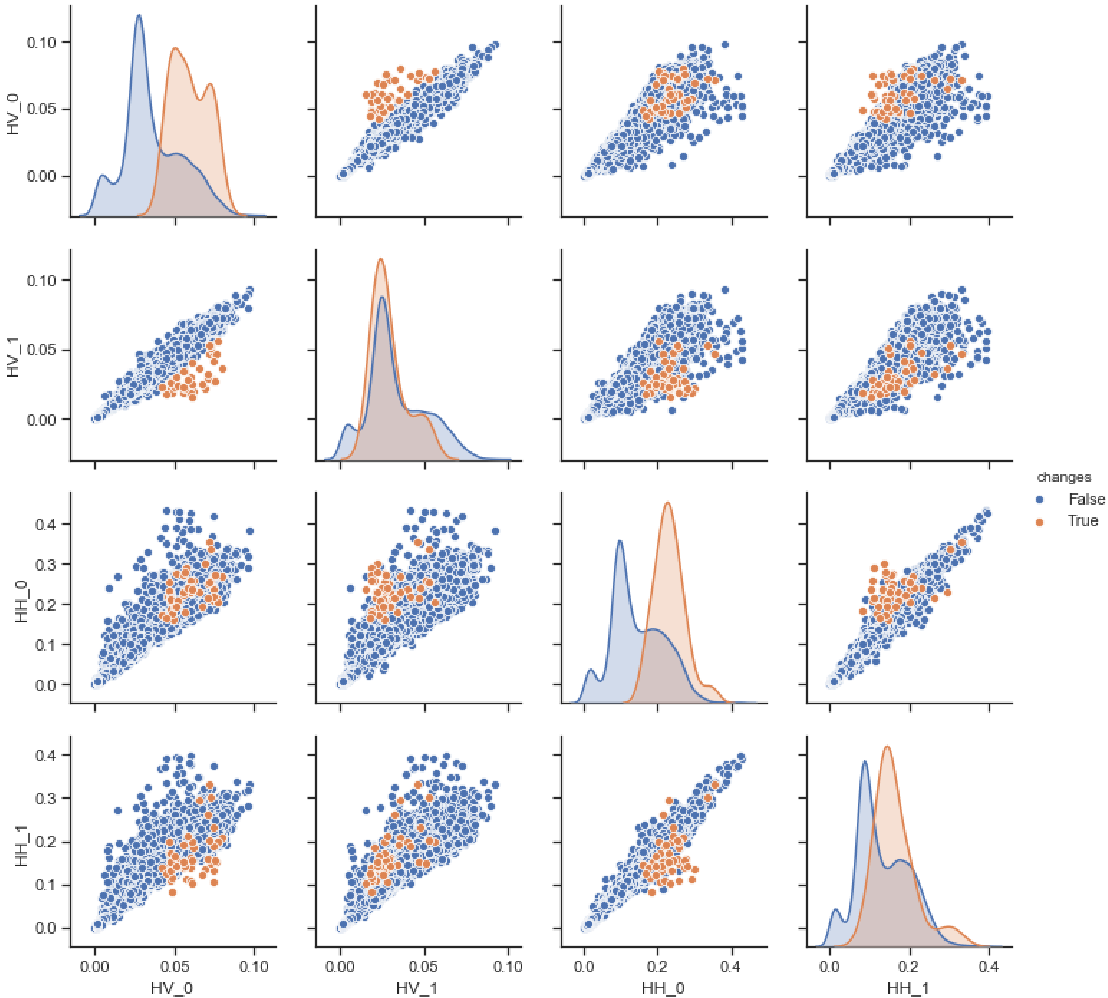
[27]:
```
df = pd.DataFrame({'HH_0': mean_features_hh_0.ravel(),
                   'HH_1': mean_features_hh_1.ravel(),
                   'HV_0': mean_features_hv_0.ravel(),
                   'HV_1': mean_features_hv_1.ravel(),
                   'changes': (class_features==2).ravel(),
                  })
df.head()
```

[27]:

	HH_0	HH_1	HV_0	HV_1	changes
0	0.194762	0.154122	0.061739	0.044976	False
1	0.207661	0.106981	0.074216	0.027814	True
2	0.224095	0.203775	0.047337	0.033440	False
3	0.197110	0.137660	0.039180	0.023662	False
4	0.224627	0.175124	0.053187	0.036412	False

[28]:
```
sns.set(style="ticks")

sns.pairplot(df,
             vars=['HV_0', 'HV_1', 'HH_0', 'HH_1'],
             hue="changes")
```

[28]: <seaborn.axisgrid.PairGrid at 0x7fb30a74f240>



7 Note

Now that you have your changed areas as shapefiles you can view them in Google Earth Pro and use the time lapse functionality (demonstrated here) to validate these areas using very high resolution imagery.

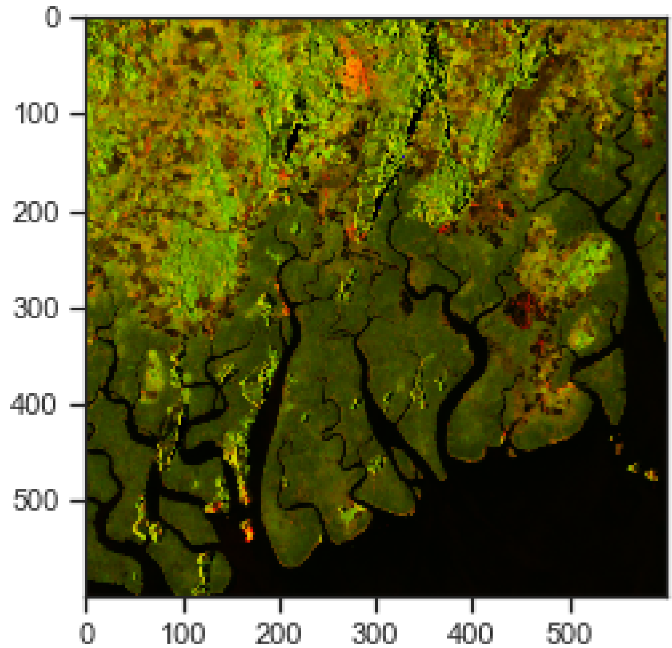
8 Appendix - RGB Composites and Mapping Mangroves with HH

Suppose you have a region that you want to identify mangroves using SAR. Let's do this using the ASF data.

[29]:
```
rgb_temp = bands_tv[:2] + [bands_tv[0] / (bands_tv[1] + .00001)]
rgb_temp = list(map(scale_img, rgb_temp))
rgb_0 = scale_img(np.stack(rgb_temp, axis=2))
```

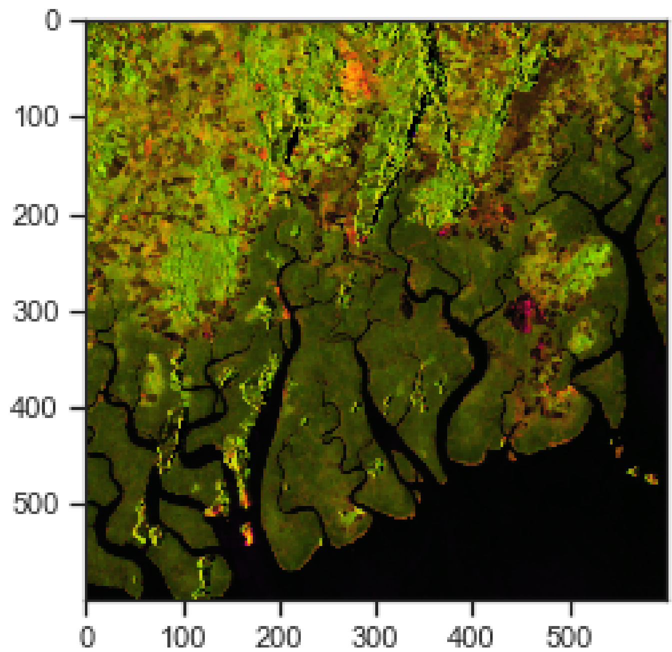
```
plt.imshow(rgb_0)
```

[29]: <matplotlib.image.AxesImage at 0x7fb30c3c89b0>



[30]:
```
rgb_temp = bands_tv[2:] + [bands_tv[2] / (bands_tv[3] + .001)]
rgb_temp = list(map(scale_img, rgb_temp))
rgb_1 = np.stack(rgb_temp, axis=2)

plt.imshow(rgb_1)
```

[30]: <matplotlib.image.AxesImage at 0x7fb2c8860c18>



[31]:
```
p = profile.copy()
p['count'] = 3

with rasterio.open('rgb_0.tif', 'w', **p) as ds:
    ds.write(rgb_0.transpose([2, 0, 1]))

with rasterio.open('rgb_1.tif', 'w', **p) as ds:
    ds.write(rgb_1.transpose([2, 0, 1]))
```

Let's classify the first image using just HH.

[32]:
```
superpixel_labels = felzenszwalb(hh_1,
                                 scale=.5,
                                 sigma=0,
                                 min_size=15,
                                 multichannel=True)
```

We'll use the standard deviation too.

[33]:
```
mean_features = get_superpixel_means_as_features(superpixel_labels, hh_1)
std_features = scale_img(get_superpixel_means_as_features(superpixel_labels,␣
 ↪hh_1))
```

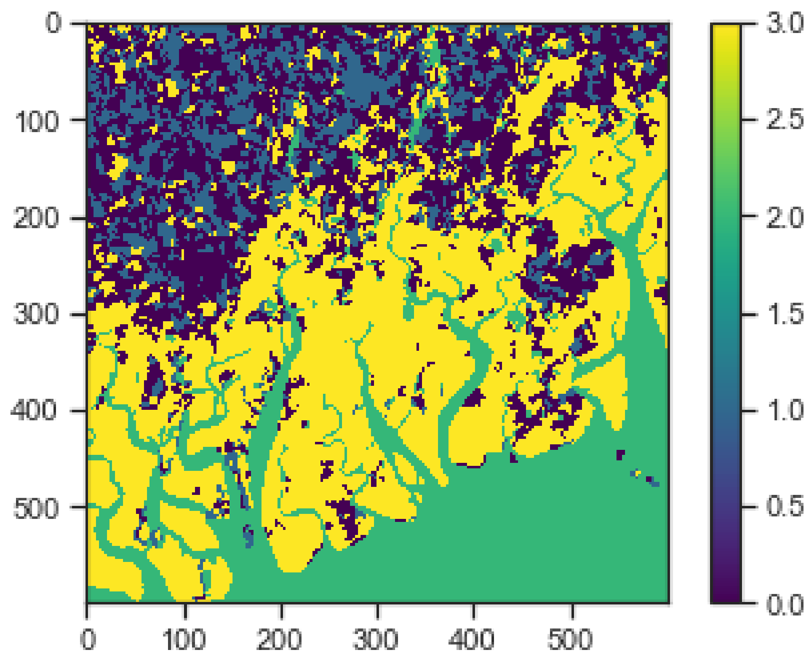
[34]:
```
## Answer

model = KMeans(n_clusters=4,
```

```
                random_state=0,
            )
X_train = np.concatenate([mean_features, std_features], axis=1)
model.fit(X_train)
```

[34]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[37]:
```
class_features = model.labels_.reshape((-1, 1))
classes = get_array_from_features(superpixel_labels, class_features)
plt.imshow(classes, cmap='viridis')
plt.colorbar()
```

[37]: <matplotlib.colorbar.Colorbar at 0x7fb2a9ee7358>



We definitely can see the mangrove areas and water correctly classified near the coast but there is some confusion as you go inland. You can Check the other areas using a basemap in QGIS with our HH layer. Since Mangroves are near the coast, you can always consider areas within some fixed distance to the coast as done here. You could also do some morphological filtering as we did earlier to remove small mangrove areas (false positives are those that are small in size).

We'll save the classes above as a tif.

[36]:
```
p['count'] =1
p['dtype'] = 'uint8'
```

```python
p['nodata'] = None
with rasterio.open('classes.tif', 'w', **p) as ds:
    ds.write(classes.astype('uint8'), 1)
```

For the mangrove areas, you may wish to compare global mangrove watch.

Remember, the model we used was k-means, one of the simplest algorithms. You can use any supervised or unsupervised classification algorithm in sklearn or convolutional neural nets as done here.

9 Acknowledgements

We gratefully and humbly acknowledge JAXA for the use of ALOS-2 data to generate a coherence image over Mondah, Gabon generated using ISCE2.

We are grateful to the ASF for providing easy access to ALOS-1 radiometrically and terrain corrected tiles over Mondah and Ko Panyi.

A portion of this research was performed at the Jet Propulsion Laboratory, California Institute of Technology. US Government Support Acknowledged.